Denis Sirbu

Software testing environment

2016

Denis Sirbu

Software testing environment

Mestrado em Engenharia Eletrónica e Telecomunicações

Trabalho efetuado sob a orientação de:
Helder Daniel

Setembro 2016

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Assinatura: _____

Data: _____

Abstract

In this thesis it is proposed a distributed software testing environment, that is able to test a wide variety of applications, such as a user space processes, kernel space processes, web applications and others. The testing environment is supported by an API of probes, where each probe has a specific test task according to its purpose. The base API can be extended to fulfil new testing requirements. This environment can be applied to general software testing, programming contests and as a support for programming classes as the Mooshak automatic judging environment. The essential differences between both these environments is where the software testing is performed. Unlike Mooshak, the proposed test environment can use client computers to do the actual test. This reduces the overhead on the server dramatically, which is especially useful when there are many test submissions simultaneously. These are the cases of classroom environments, lab environments or programming contests. Another option is to have a set of testing computers, or slaves, ready to test user code. However this way more hardware is required. The only requirements for a computer to be a slave, part of the testing environment, is that is has installed a java client application that communicates with the master computer also addressed here as the main server. On the main server a portal allows users to access this testing environment. This master computer is also responsible to distribute the testing workload, according to the choosen strategy, sending to each slave the executable and testing probes, which includes the matching pairs of inputs and expected outputs. After the slaves had performed the tests, they generate a report with information on the tests, and send it back to the master, being available to the users on the portal. To support simultaneous clients the portal is thread based, being launched a new thread to serve each new client connection. Communication between all computers in the test environment, is done using the BSD sockets API.

Keywords.

Automatic Software testing environment; Distributed environments; Object oriented programming; Web applications; Java.

Resumo

Hoje em dia é muito útil ter uma boa ferramenta de teste de software para quem quer fazer um programa com especificações restritas. Isto pode custar uma fortuna, e por sua vez não significa que a compra deste tipo de serviço garanta que o programa desenvolvido sob sua especificação será 100 % livre em termos de erros, bugs e até mesmo falhas gerais. Estas 3 palavras que nenhuma empresa quer ouvir. Na historia da informática aconteceram imensas falhas que poderiam ser evitadas testando melhor os programas. Por um lado o teste de aplicações torna-se mais fácil hoje em dia, se as APIs utilizadas já tiverem sido sujeitas a testes e livre de erros, proporcionando dessa forma intrínsecas rotinas que podem ser incorporadas em aplicações sem a necessidade do programador as reescrever. Por outro lado o teste dessas APIs deve ser extenso e minucioso. Desta forma uma interface gráfica de utilizador (GUI), por exemplo, pode ser construída a partir de bibliotecas de componentes adequadas à linguagem escolhida para o desenvolvimento. Uma vez que estes componentes são objectos pré-programados que foram depurados e testados anteriormente, a necessidade de testá-los novamente não é necessária, sendo apenas necessário testar a sua integração com a aplicação. Assim o teste de software é um processo, ou uma série de processos, destinado a garantir que o código respeitas as especificações e que não apresenta comportamentos não intencionais, mas sim previsíveis e consistentes, oferecendo o melhor desempenho sem surpresas para os utilizadores. Para efetuar testes podemos ter duas opções: Analisar o código, referidos como testes de caixa branca, ou ignorar o código e tratar o sistema como uma caixa negra. Neste conceito, que é escolhido para o ambiente de testes proposto aqui, descrito aqui, são aplicadas ao sistema a testar um conjunto de entradas e verificado se para cada entrada a saída é igual à saída esperada de acordo com as especificações. Este tipo de testes é usado vulgarmente, mesmo em concursos de programação, sendo um dos exemplos os concursos organizados pela ACM-ICPC (International Collegiate Programming Contest), para equipas de estudantes. Estas competições locais, nacionais e internacionais, são direcionadas para equipas que consistem em cerca de 3 estudantes de universidades ou outras instituições. O objetivo consiste em escrever programas que para entradas dadas irão gerar as saídas

de acordo com as especificações. Concursos semelhantes são adotados em muitas universidades, por causa do aspeto competitivo, que faz com que os alunos tenham um objetivo ao completar um exercício. Os testes de caixa negra podem ser adaptados em varias fases de construção de um programa. Podem ser usados nos blocos básicos de construção, onde os componentes são testados um a um, com determinadas entradas e saídas esperadas. Numa fase posterior de desenvolvimento de um sistema podem ser construídos testes de integração, também de caixa negra, onde os componentes que foram testados anteriormente são agora testados em termos de integração. Podem também ser aplicados a módulos e a toda aplicação. O objetivo principal desta dissertação foi a criação de um ambiente de teste de software automático e distribuído, que possa ser usado nos casos acima descritos. Este ambiente foi projetado para poder ser utilizado numa grande variedade de aplicações, tais como: processos em espaço do utilizador, processos em espaço do núcleo, aplicações web e outros. Os testes são implementados com suporte de uma API de sondas, em que cada tipo tem uma tarefa específica de teste de acordo com o que se pretende testar. Cada sonda inclui também um conjunto de entradas e saídas esperadas. Para efetuar diversos testes sobre uma aplicação, o utilizador escolhe uma coleção de sondas, dos tipos apropriados ao que pretende testar, define nelas as entradas e saídas esperadas e envia-as junto com o executável para o ambiente de testes, que as aplica uma a uma e gera um relatório dos testes. Esta API foi projetada como uma hierarquia de classes, onde classes bases permitem doar comportamento elementar, de modo que aos tipos de sondas já definidos podem ser adicionados novos pelo utilizador. A API está escrita na linguagem de programação Java. Este ambiente de teste é em certa medida semelhante ao ambiente de teste utilizado atualmente em ambiente de sala de aula de algumas universidades e também no Torneio Interuniversitário de Programação, chamado Mooshak. As diferenças essenciais entre ambos os ambientes consistem onde são efetuados os testes. No ambiente aqui proposto, tal como no Mooshak, os testes podem ser efetuado num ou mais servidores a que os clientes se ligam. Mas ao contrário do Mooshak, este ambiente pode usar também o computador do cliente que está a efetuar o pedido de execução do teste. Isto reduz a sobrecarga no servidor dramaticamente, especialmente quando existem muitas execuções de testes em simultâneo. A situação descrita pode acontecer em ambiente de sala de aula, ambiente de laboratório ou concursos de programação. Em qualquer das arquiteturas propostas existirá um conjunto de computadores prontos para testar o código dos utilizadores. Nestes, o ambiente de teste poderá ser implementadas em máquinas virtuais, o que permite aumentar a segurança ao testar código desconhecido que possa ser perigoso ou maligno. Os únicos requisitos necessários para que um computador integre o ambiente de teste consiste em ter instalado um programa java que comunicará com o servidor de testes principal. Deste modo temos uma arquitetura master-slave. A interacção com os clientes, utilizadores, é efetuada através de um portal executado num computador denominado servidor. Este

computador é também responsável por distribuir a carga de testes, de acordo com a estratégia escolhida, enviando para cada slave o executável a testar e a coleção de sondas. Após o teste cada slave cria um relatório dependendo dos ensaios realizados que envia de volta para o master, sendo disponibilizado ao utilizador no portal. Para suportar serviço de clientes em simultâneo, para cada ligação cliente o portal cria uma nova thread dedicada ao seu serviço. A comunicação entre todos os computadores no ambiente de testes é efetuada utilizando a API de soctes BSD.

Termos chave.

Teste de programas automático; Ambiente distribuído; programação orientada a objectos; Aplicações Web; Java.

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported me throughout this long MIEET program. I am thankful for their aspiring guidance, invaluable constructive criticism, and friendly advice during the project work and during my university life. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues I had during this path.

I express my warm thanks with love to my parents and my friends for their support and guidance at the University of Algarve.

I would also like to extremely thank my project supervisor, teacher and guide - Helder Daniel, and all the people who provided me with the facilities being required and conducive conditions for my master thesis.

Contents

List of Figures 9

1 Introduction 10
- 1.1 Objectives . . . 10
- 1.2 History . . . 12
  - 1.2.1 The Ariane 5 rocket . . . 12
  - 1.2.2 The Y2K . . . 12
  - 1.2.3 USA "Star Wars" program . . . 13
  - 1.2.4 Collapse in ambulance system of London. . . . 13
- 1.3 What and how to test . . . 15
- 1.4 ICPC . . . 16
- 1.5 Structure of this work . . . 17

2 Background information and theory 18
- 2.1 State of art . . . 18
  - 2.1.1 Software life cycle model . . . 20
    - 2.1.1.1 Pre-alpha . . . 21
    - 2.1.1.2 Alpha . . . 22
    - 2.1.1.3 Beta . . . 22
    - 2.1.1.4 Release . . . 22
    - 2.1.1.5 Support . . . 23
    - 2.1.1.6 End-of-Life . . . 23
  - 2.1.2 V-shaped software life cycle model . . . 24
    - 2.1.2.1 Component test . . . 25
    - 2.1.2.2 Integration Test . . . 25
    - 2.1.2.3 System test . . . 25
    - 2.1.2.4 Acceptance Test . . . 26
- 2.2 Definition . . . 26
- 2.3 Technical workflow diagram . . . 30
  - 2.3.1 Client-side interface . . . 30
  - 2.3.2 Server-side interface . . . 32
- 2.4 Programming languages . . . 34
  - 2.4.1 PHP , http, html . . . 34
  - 2.4.2 Tomcat server . . . 34
  - 2.4.3 Why Java? . . . 35
  - 2.4.4 JSP . . . 35
  - 2.4.5 MVC form . . . 36

2.4.6 Http session . . . 37
2.4.7 Java Servlets. . . . 38
2.4.8 Master-slave concept . . . 38

3 Experimental setup **39**
3.1 Web server . . . 39
3.2 Supplemental computer . . . 40
3.2.1 Communication through sockets . . . 40
3.3 Coding phase. . . . 41
3.3.1 GET and POST . . . 42
3.3.1.1 HttpServletResponse object . . . 44
3.3.1.2 HttpServletRequest object . . . 46
3.3.2 Java server pages . . . 48
3.3.2.1 JSP life cycle . . . 49

4 Software testing **53**
4.1 Implementation of the Software testing environment . . . 54
4.1.1 Work flow . . . 54
4.1.2 User contribution . . . 56
4.1.3 Server as itself . . . 57
4.2 Before the execution of the Software testing environment . . . 58

5 Case Studies **60**
5.1 Case study 1 . . . 62
5.1.1 ProbeAPI . . . 62
5.1.1.1 Probe. . . . 63
5.1.1.2 TestProbe . . . 63
5.1.1.3 TestCase . . . 64
5.1.2 Case study 1 - Network probe . . . 66
5.1.2.1 TestProbeNet . . . 66
5.1.2.2 TestProbePing . . . 68
5.1.2.3 TestProbeSocket . . . 69
5.1.3 Case study 2 - Userspace probe . . . 71
5.1.3.1 TestProbeUserSpace . . . 71
5.1.3.2 TestProbeUSstdinout . . . 72
5.1.3.3 TestProbeUserSpace . . . 73
5.2 The use of the testing environment to aid computing students exercises . . . 74

6 Conclusions and future work **80**
6.1 Recalling the initial part of the work . . . 80
6.2 Future work . . . 81
6.3 Conclusion . . . 82

Bibliography **85**

List of Figures

2.1 Waterfall Software life cycle model 20
2.2 V shape model 24
2.3 Programmer side design 31
2.4 Web-Server Design 33
2.5 CGI 36
2.6 MVC 37

3.1 Http servlet 42
3.2 Httpresponse 45
3.3 JSP 49

4.1 MVC interaction 54
4.2 Testing unit 56
4.3 Software Testing scheme 57

5.1 Probe UML. 61
5.2 Fibonacci practical experience 75
5.3 Test of logging to greetings page(29 of 100 get requests). 77
5.4 Test of logging to greetings page(29 of 600 get requests). 77
5.5 Test of logging into server(29 of 100 post requests). 78
5.6 Test of logging into server(29 of 600 post requests). 79

6.1 Comparison between 2 samples 84

Chapter 1

Introduction

1.1 Objectives

The main objective of this thesis is to create an automated and distributed software testing environment which must be able to test various types of applications, such as user space processes, kernel space processes, web applications, etc.
The tests were implemented as a black box method testing where each case consists of a pair - input / expected output.
The testing process is implemented in an API of probes, where each probe has a specific test purpose. The base API can be extended to fulfil user requirements and the testing environment should be supported by this API.

First of all, it is planned to make an environment that will work in a distributed computer system, also it could eventually test a wide variety of applications. The reason that it was chosen a black-box type testing it is because of the simple and careful input procedure which can be easily adapted for any kind of desired output cases. The testing process is implemented as an API of probes, where each probe will have its own testing objective.

Probe is an object written in java and it consists essentially from some process which will give to us a testing case. To facilitate the creation of new probes it is described and exemplified in chapter 5 some of them that already exist.

The goals that are intended to be achieved with this work are:

1. Software testing programmed in Java, which reasons are: its object oriented features(for more details - [1]), the wide range of developers that use it so in future if I would like to use this work anyone knowing java could understand it and mainly previous experience as a developer in this type of programming language.

2. The environment as program is distributed between its processes (threading), and if required, between other computers that take a part of testing environment (BSD sockets).

3. The testing environment is shown working, presenting 2 study cases.

4. This implementation is modelled and documented to be easily understandable and expandable for the future work, and its documentation will consist primarily of this work that describes the environment.

1.2 History

According to [2], the big start of spending time on software testing environments started in 1970, because of big progress in computers eventually made among this year. Nowadays software testing is more and more difficult because of variation of programming languages, operating systems, and hardware platforms that have evolved. Each year it is possible to have more and more powerful computers that make instant calculations instead of hours like some years ago, providing us - programmers with very good weapons.

A major problem of the software industry is its inability to develop error-free software. Had software developers ever been asked to certify that the software developed is error-free, no software would have ever been released. Hence "software crises" has become a feature of everyday life with many well-publicized failures that have had not only major economic impact but also have become the cause of loss of life. In history, we can see some of this cases.

1.2.1 The Ariane 5 rocket

According to [3], it was designed by European Space Agency and it was launched on June 4, 1996. It was an unmanned rocket and unfortunately exploded only after 40 seconds of its take off from Kourou, French Guiana. The design and development took ten long years with a cost of 7 billion of US dollars. An enquiry board was constituted to find the reasons for the explosion. The board identified the cause and mentioned in its report that: “The failure of the Ariane 5 was caused by the complete loss of guidance and altitude information, 37 seconds after the start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system. A software fault in the inertial reference system was identified as a reason for the explosion by the enquiry committee. The inertial reference system of the rocket had tried to convert 64-bit floating point number of horizontal velocity to a 16-bit signed integer. However, the number was greater than 32,767 (beyond the permissible limit of 16-bit machine) and the conversion process failed.

1.2.2 The Y2K

This problem was the most critical problem of the last century. The whole world was expecting something drastic on January 1, 2000. Significant sums of money were spent by software companies to get rid of this problem. What was the problem? It was simply the case of using two digits for the year instead of four digits. For instance, 1965 was

considered as 65. The developers could not imagine the problem of the year 2000. What would happen on January 1, 2000? The last two digits i.e. 00 may belong to any century like 1800, 1900, 2000, 2100, etc. The simple ignorance or a faulty design decision to use only the last two digits of the year resulted from the serious Y2K problem. Most of the software was re-tested and modified or discarded, depending on the situation. (Taken from [4])

1.2.3 USA "Star Wars" program

"Patriot missile" was the result of this USA program. This missile was used for the first time in the Gulf war against the Scud missile of Iraq. Surprisingly, "Patriot missiles" failed many times to hit the targeted Scud missile. One of the failures killed 28 American soldiers in Dhahran, Saudi Arabia. An investigation team was constituted to identify the cause of failure. This failure was a software fault, there was a slight timing error in the system's clock after 14 hours of its operation. Hence, the tracking system was not accurate after 14 hours of operations and at the time of the Dhahran attack, the system had already operated for more than 100 hours. (Taken from [5])

1.2.4 Collapse in ambulance system of London.

The software controlling the ambulance dispatch system of London collapsed on October 26-27, 1992 and also on November 4, 1992, due to software failures. The system was introduced on October 26, 1992. The London Ambulance Service was a challenging task that used to cover an area of 600 square miles and handled 1500 emergency calls per day. Due to such a failure, there was a partial or no ambulance cover for many hours. The position of the vehicles was incorrectly recorded and multiple vehicles were sent to the same location. Everywhere people were searching for an ambulance and nobody knew the reason for non-arrival of ambulances at the desired sites. The repair cost was estimated to be £9m, but it is believed that twenty lives could have been saved if this failure had not occurred. The enquiry committee clearly pointed out the administrative negligence and over-reliance on "cosy assurances" of the software company. The administration was allowed to use this system without proper alternative systems in case of any failure. The committee also termed the possible cause of failure as [6], [7]: "When the system went live, it could not cope with the volume of calls and broke under the strain. The transition to a backup computer system had not been properly rehearsed and also failed." (Taken from [8])

Software testing became easier nowadays, because the array of software and operating systems is much more sophisticated than ever, providing intrinsic well-tested routines

that can be incorporated into applications without the need for a programmer to develop them from scratch. Graphical user interfaces (GUIs), for example, can be built from a development language's libraries, and, since they are preprogrammed objects that have been debugged and tested previously, the need for testing them as part of a custom application is much reduced. Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended. Software should be predictable and consistent, offering no surprises to users.

1.3 What and how to test

Having here enough explanation about failures that happened in the history we can conclude that testing of the programs is very important. To start the main part of this work - doing tests, we should not expect that there will be necessarily the understanding of what program or software does, and the principle that is described here is followed as a black box testing. So mainly if we have input we should have a certain output, the test becomes successful if its desired output would be same as inserted input-output case. In literature there are some terms that we should describe, in standard - IEEE 610.12 [9], the term **failure** means an unexpected crash of a program or wrong output; a **fault** causes a failure, a good synonym for this term would be defect or internal error; also we have an **error** that is a cause of a fault or defect, normally it is a mistake of a person, and it is possible to find more errors if a person is under pressure for a long time. Looking back to our main part of the work this black box "study" is the main type of tests that a software tester uses to make over's software "broken" - with defects and consequently with failures. But what happens when we detect something suspicious, how would we correct them?

To be able to correct a defect or bug, essentially it must be localised in the software. We know the consequence of this defect but not the precise location in the software and localisations and corrections of defects is the task for a software developer and this is called as **debugging**. Repairing a defect generally increases the **quality** of the product because the **change** in most cases theoretically it does not introduce new defects. However theory is always different from real cases, and in practice, correcting defects often add one or more new defects. The new defects may then introduce failures for new, totally different inputs which is called as masking defect. Such unwanted side effects make testing more difficult. The result is that we must not only repeat the **test** cases that have detected the defect but also provide even more complex test cases to detect possible side effects.

We can't misunderstand that debugging is not the same thing as testing if we will think about it, to debug, we should be very familiarised with the code to understand it perfectly. And generally debug is the task of localising and correcting faults, and at another side, testing is a systematic detection of failures which will indicate the presence of a defect.

A test run or test suite includes execution of one or more test cases. A test case contains defined test conditions. In most cases, these are the preconditions for execution, the inputs, and the expected outputs or the expected behaviour of the test object. A test case should have a high probability of revealing previously unknown faults.

1.4 ICPC

ICPC - International Collegiate Programming Contest is where the automated software testing environments are widely used. This is a world contests organisation where participants are divided into competing teams which consist of more or less of 3 students from universities, and its objective is to write a program that will generate a certain output after introducing some input cases. This kind of concept is adopted in many universities and because of competitive aspect, this makes students have an objective while completing an exercise. For this purpose was started a project in the University of Porto that was called Mooshak - the software testing program, is available for download from [10]. It is written in TCL language and its architecture consists of the web server as "user-GUI" and at same time, this web server makes all of the tests. Users, in this case - students, upload their code while the testing environment is prepared by the teacher as automated judging system, and there can be restricted the date limit when will be opened this kind of context. This Mooshak system can only be installed on the sequence of next requirements:

1. Linux based operative system.
2. Apache 1.2 version installed.
3. TCL 8.3 version installed.
4. Compilers for the creation of an executable.

1.5 Structure of this work

This is the chapter 1 of introduction and it is a big entrance to how is it difficult to create an error free software and what stages we need to pass to have it designed as an entire "system", as it is shown a very important thing described which is testing mechanism and its importance as software developing tool.

In next chapter, Chapter 2, it is talked about background information that it was needed to analyse in what programming language the environment is written and what kind of services already exist to do the main part of the work - tests.

Testing environment is separated into two parts the first one is web interface as GUI, and the second is the testing unit. It can be implemented in 2 separate systems.

Chapter 3 is experimental setup of web interface and this testing unit and it is about how was created the testing environment and what is the main purpose of it with all it's system motor functions that were written during the work.

In chapter 4 it is explained the software tests with probes that are already implemented in the API.

Chapter 5 exemplifies the probes and their mechanism that is needed to understand for generating new testing probes. A probe is a facility for implementation of testing environment part which is done in experimental setup that provides us with some data that can be analysed for constructing of conclusions.

And finally, the last but not the least chapter 6, is the conclusion and future work.

Chapter 2

Background information and theory

2.1 State of art

Nowadays it is very useful to have a good software tester for someone who wants to make a program with restrict specifications. Sometimes it can cost a lot of money to have this kind of services in its owners firm or in a case of some freelancer that is working from home. But paying money don't guarantee that this tester can't fail and miss some hidden error, and for this purpose, there are many solutions that are possible to find on the internet, but sometimes these solutions are not confidential, or if so are very expensive.

Imagine that you put your code exposed on a forum where many programmers are involved and it is for free getting a suggestion about how to resolve some difficult part that you are passing through, but putting your idea on internet forum it is risky, because certainly someone will try to use your part of the work in his own way and who knows even earn money on it. In the market, we can easily find a lot of "beta-testers" as users that are participating in this kind of work.

A good example of where this testing is very used is in video games. A company that made a video game, gives its "beta" version for some user that enjoyed the story and wants to be the first one who plays it, but this kind of testing is not only done with video games. And to prove these let's see the next example when Google launched its product - Google-glasses they gave the first version of a pair to beta-tester who was first who registered in the process of marketing and had a very good historical recommendation. It is always done so with a purpose to see if all defects were fixed and not only, another purpose is for people who play the game (in case of previously said video games production) will receive the first version of this kind of product but

in exchange they should make a detailed report of the reliability of it and if they find something that is for their opinion is out of sense they should carefully explain the whole situation. This is the steps of the contract that beta-testers accept with any company. Examples of this contracts you can see in the next sources: [11], [12], [13].

Coming back to the principles of this work we should not forget to explain what is a program. A program is any set of instructions that are executed by a machine. It can be a very simple sum that even a calculator can do to a very hard code that an operating system has in it, such as a driver which makes your webcam capture the image or USB interaction. Technically a program is almost the same as software but with a difference that software can consist of a program and some database for example. Note, in this document it is considered that program and software is the same thing.

2.1.1 Software life cycle model

According to [14], about what is a software testing we first of all shall start from software release which consists from the next stages: **pre-alpha**, **alpha**, **beta** (gamma, delta... if necessary), **release**, **support** and **End-of-life**. Reminding the video game example - the beta-testing, that was described in previous introductory part of State of art, says that it takes part of beta stage, and also lets see as the name of subsection says the software life cycle, where are described all stages of constructing a fully functional program and see what work has consisted inside of each of stage.

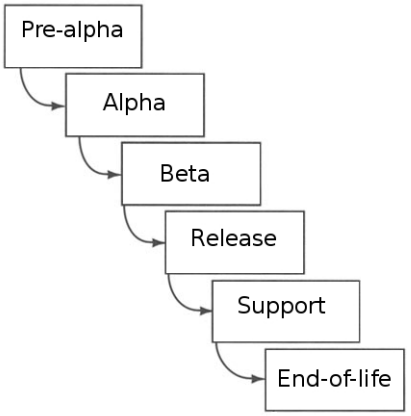


FIGURE 2.1: Waterfall Software life cycle model

Next, is shown in detailed information of all the stages that are behind of a successful software production, this doesn't mean that all companies are using this method (figure 2.1), but at least the consulted sources do [14] [15]. And a big example of this is Microsoft Company when they created its Internet Explorer they used something similar, this can be confirmed in paper [2]. Coming back next it is explained each of stages of this waterfall model.

2.1.1.1 Pre-alpha

This stage is all that is performed before testing, more specifically - planning and writing the engine functions, that make the software work, basically if the software is in its pre-alpha stage it means that not everything is done yet and also all the functionality might not be implemented yet or may be unstable. For this reason, we can't start to test it yet. This stage follows the next "thinking elements" that are **requirements analysis** and **software design**.

As [16] says main objectives of **requirements analysis** are to detect and resolve conflicts between requirements and elaborate system requirements to derive software requirements. Basically, here we create our goals, and all these steps are made to have a real look of the idea how much work is needed to get to these goals. Having our requirement analysis done we also can estimate the cost of this work. Requirements analysis is also divided in various stages like Requirements Classification, Conceptual Modelling, Architectural Design and Requirements Allocation, Requirements Negotiation and Formal Analysis. It is assumed as **software design** a complete sequence of goals that should be achieved and each of them are main. Normally software engineers use requirements analysis to construct these goals using some techniques which consist of this software design, like UML.

Software developing is the main process of this phase because during this process are created functions that in future make our software work. This is accomplished by analysing software design, and bringing it from paper to code. **Unit testing** is a process where are tested individual units of source code, sets of one or more computer program modules together with associated control data. A unit of source code should be considered as the smallest testable part of an application. If the code is written with care a unit is a function which can be easily tested.

After all, these procedures starting from requiring analysis and finishing with unit testing we can pass to next phase - Alpha phase.

2.1.1.2 Alpha

This is the first phase of a complete software testing, it is Alpha because it starts as the first letter in Greek alphabet. Software programmers at this stage start to perform white box testing - that tests internal structures or workings of an application. To validate a set of tests it is performed a black box testing. Because of many varieties of software in this work, here it is where is used exactly the black box testing, where we know the input and expected output if it is not equal than the program has errors. Also if the program passed white box testing it means that the program is functional and it is called Alpha release. After black box testing, we can gather all feedback and add the possibility of new features. So these black box tests are fundamental in software building.

Imagine now that we have a company that has to build a program with its pre-alpha phase accomplished, this company has 4 teams as software engineers and each of them does this work, to control if every team had proceeded to the initial goal, they should pass the black box test with success where none of each team do know the input and output.

2.1.1.3 Beta

As the second letter of the Greek alphabet is Beta that should mean that software must be functional and must be also usable, these does not grant that it will not have any bugs and sometimes there are bugs that are already known.

The reason that Beta phase appears in the waterfall diagram is because it is not only important to have it but with its help we can understand where and how the program works incorrectly. And as it was mentioned before many companies give this version of "faulty" software to customers and then, they - customers, become beta-testers. The function of each beta-tester is to make a report to a company if they find any anomalies, and some company's pay them for this kind of services, but now these tend to be volunteer services, free of charge and instead they often receive versions of the product they test, discounts on the release version, or other incentives.

2.1.1.4 Release

When all mistakes are fixed it is launched a successful and bug-less the final version of software, normally it comes with RTM or RTW tag. RTM means "release to manufacturing" and it is ready to be delivered or provided to the customer, and RTW - "release to web" it is basically software delivery that utilizes the Internet for distribution, no

physical media are produced in this type of release mechanism by the manufacturer, since the internet is getting all over the world RTW is becoming more and more usable.

2.1.1.5 Support

During the supported lifetime, software is sometimes subjected to service releases or service packs. For example, Microsoft has so known service packs for the 32-bit editions of Windows XP and two service packs for the 64-bit editions. Such service releases contain a collection of updates, fixes, and enhancements, delivered in the form of a single installable package. Some software is released with the expectation of regular support. Classes of software that generally involve usual support as the norm include anti-virus programs and massively multiplayer on-line games. A good example of a game that utilizes this process is Minecraft, an indie game developed by Mojang, which features regular "updates" featuring new content and bug fixes.

2.1.1.6 End-of-Life

When software is no longer sold or supported, the product is said to have reached end-of-life, to be discontinued, retired, or obsolete, but user loyalty may continue its existence for some time, even long after its platform was obsolete.

These all steps are normally done for building a successful software, but in software testing books like [17] [2] is used another form of the model, and to understand why it is made we should try to prevent serious mess at the beginning and not to wait until the final steps are done and sell the software as complete one.

2.1.2 V-shaped software life cycle model

The V-shaped model is the modified form of the waterfall model with a special focus on testing activities. This model focusses exactly only on Pre-Alpha and Alpha stages, because only this phases treat the program on its acceptance and capability of programming code with desired objectives that were put initially.

It is made as V-shape because we need to understand that developing and testing are things of the same relevance. On figure 2.2 is exemplified a scheme of this model, where left branch corresponds to developing and right branch corresponds to the testing process which both interacts with each over. Since there are a lot of different V models, I've chosen this one as a condition of most recent published date of all books I read.

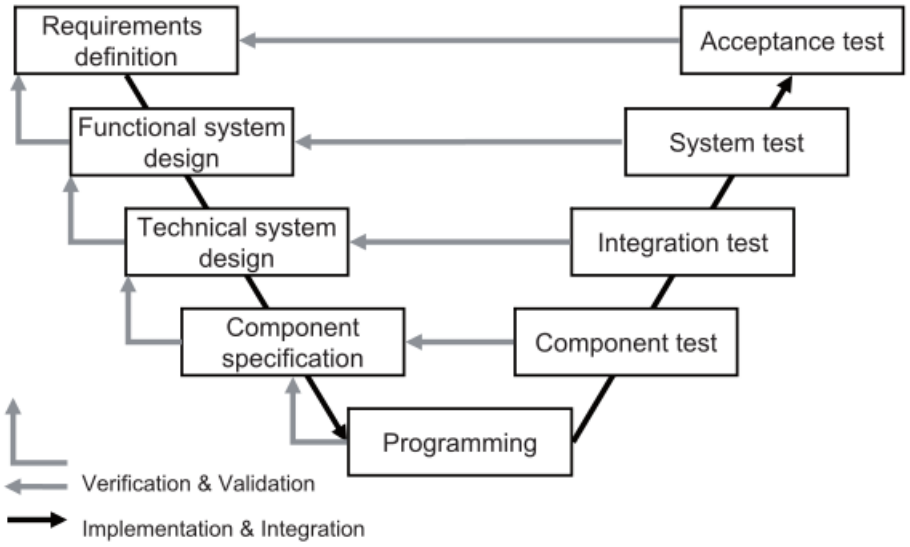


FIGURE 2.2: We can see similar phases between pre-alpha stage and this V model. Requirements definition - Requirements analysis; Functional and Technical system design, Component specification - Software design; Programming - Software developing. Image was taken from book [17]

Also, this model seemed to be more appropriated to compare with. Let's start analysing it then. So as it is mentioned in the description **Requirements definition** is same as **requirements analysis**, okay then this step is same as in the previous model, then let's remind what was said before - detect and resolve conflicts between this requirement and elaborate system requirements to drive to software requirements, lets add here the word customer, if we look at the right branch of the figure 2.2 we can see the corresponding to every phase its testing specifications. **Component test** - verifies whether each software component correctly fulfils its specification. **Integration test** - checks if groups of components interact in the way that is specified by the technical system design.**System**

test - verifies whether the whole system meets the specified requirements. **Acceptance test**, this test checks if the system meets the customer requirements, as it is specified in the contract and/or if the system meets user needs and expectations.

2.1.2.1 Component test

As the first level testing, these tests are done how the title says by components. It certainly depends on the programming language used to write the software, if it is an objective oriented language it tests classes, otherwise, it is separated by modules or units. Basically, these tests are same as unit tests described before in pre-alpha explanation. And this tests should be prepared for each unit/module separately to avoid influences of other components. Component testing is very closely related to development. Here tester should have access to the source code, doing so this code is analysed and it is created appropriated functions-testing components, also it is done by black box testing.

2.1.2.2 Integration Test

As you know from figure 2.2 the second level of the V-model is the integration test. It is required that all component tests must be accomplished, and all defects must be corrected. Integration test as you can understand from the name is an integration of all units, this means that now we can perform a kind of group testing, combining these components and composing so some larger structural units. After joining all components we need to see if everything is working. Normally this integration is attended by a test manager if it is a group programming he is who controls this process controlling and making the best effort for an optimal integration strategy of the project. Also, it is supposed that test manager should have a good knowledge of strategies in how to make better integration.

2.1.2.3 System test

After the integration test is completed, the third and next test level is the system test. System testing checks if the integrated product meets the specified requirements. Test levels that we should have executed before where done against technical specifications, from the technical perspective of the software producer. But all tests should have been validated with no matter if the requirements were all implemented or not. Main functions and system characteristics result from the interaction of all system components, so they are visible only when the entire system is present and can be observed and tested only

there. The goal of the system test is to validate whether the complete system meets the specified functional and non-functional requirements and how well it does that.

2.1.2.4 Acceptance Test

Finally, the last step of this tests is Acceptance Test. The test to base on can be any document describing the system from the user or customer viewpoint, such as, for example: user or system requirements, use cases, business processes, risk analyses, user process descriptions, forms, reports, and laws and regulations as well as descriptions of maintenance and system administration rules and processes. The test criteria are the acceptance criteria determined in the development contract. Therefore, these criteria must be stated as unambiguously as possible. Additionally, conformance to any governmental, legal, or safety regulations must be addressed here.

After acceptance test, if something is wrong we downgrade to lower levels and start tests from the beginning. When this is all done and everything is working good we can pass to the next stages that are alpha and beta, and then release that already was explained previously.

2.2 Definition

This project has some similarities with Mooshak judging system. As the paper [18] says, Mooshak is a web-system that has a context management and at the same time, it is an automatic judge for some programming contexts. It becomes very useful for testers, because of the possibility of configuring the whole system as a black box and having this testing available on-line makes it perfect while competing in a team or individually.

Nowadays, Mooshak is used in programming courses and serves as a good testing system that helps the programmer to perform all kind of trials. To test, you should upload the source code and it is at same way important because it gives also to the tester an interaction, during which procedure that involves some partial evaluation, this means that when the first testing case scenario passes and the next one will fail, the programmer will not only think to rectify the program but also will try not to fail again, while searching for other unexpected outputs.

Again by referring to the paper [18], it is said that Mooshak - the web platform, is written in HTML 4.0 and all the system as itself is runnable only on Linux with Apache and all system logical functions are written in TCL 8.3 (Tool command language).

TCL

Citing from [19] - "Tcl, or Tool Command Language, is an open-source dynamic programming language. For decades it has served as an essential component in systems in organisations ranging from NASA to Cisco Systems, the source says that it is a must-know language in the fields of EDA[1], and powers companies such as FlightAware and COMPANY: F5 Networks. Along with its C library, it provides an ideal cross-platform development framework for any programming project.

Tcl is one of the only languages that is convenient for both the smallest and largest programming tasks, without ever having to ask, "Is it overkill to use Tcl here", or "If this code gets bigger, will Tcl scale?". Whenever a shell script might be used, Tcl could be used just as easily. On the other hand, entire web ecosystems and mission-critical control and testing systems are also written in Tcl. Tcl excels in all these roles due to the minimal syntax of the language, the unique programming paradigm exposed at the script level, and the careful engineering that has gone into the design of the Tcl internals.

Tcl is the least prescriptive of the scripting languages. Its syntax is described in just a dozen rules. There are no reserved words and there is no built-in syntax for control structures or conditionals. Instead, control and branching operations are, like everything else in a Tcl program, implemented as commands. For example, if is a command that in its simplest form takes a conditional expression to test, and a block of code to be executed if the condition is true. A Tcl script could replace if with its own implementation. Each program is written in Tcl essentially becomes a domain-specific language for whatever it is that program is designed to do. Tcl is to script what Forth is to stack machines: A starkly-minimal language that fits its domain like a glove." Because of the interest in java language and not in TCL this was an introduction to the reader about TCL, for more information it can be considerable the reading of the next references: [20], [21], [22].

During the investigation of the state of art there were found other software testing systems like Mooshak for example SPOJ (Sphere Online Judge), UVA (Universidad de Valladolid Online Judge), CodeChef, URIonlinejudge, a2oj and others.

[1] Electronic(s) design automation

Sphere Online Judge (SPOJ)

[23] explains that the SPOJ platform is centered around an online judge system, which serves for the automatic assessment of user-submitted programs. Some of its most important features include:

- support for more than 45 programming languages and compilers, including C, C++, Pascal, JAVA, C#, Perl, Python, Ruby, Haskell, Ocaml, and esoteric languages.
- a rapidly growing problem-set of about 13000 tasks available for practice 24hours/-day (in English, Polish, Vietnamese, Portuguese and other languages), including many original tasks prepared by the community of expert problemsetters associated with the project.
- a flexible testing system, supporting dynamic interaction with submitted programs and highly customisable output of assessment results.
- intuitive browser-based content management which allows users to set up their own contests within a matter of minutes, and to make use of tasks already available in the system.
- more than 2400 contests hosted as of 2012, ranging from instant challenges to long-term e-learning courses.

CodeCheff

Citting from [24], CodeChef is a not-for-profit educational initiative by Directi, an Indian software company. It is a global programming community that fosters learning and friendly competition, built on top of the world's largest competitive programming platform. We have also built a large community of problem curators. CodeChef was created as a platform to help programmers make it big in the world of algorithms, computer programming and programming contests. We host three featured contests every month and give away prizes and goodies to the winners as encouragement. Apart from this, the platform is open to the entire programming community to host their own contests. Major institutions and organizations across the globe use our platform to host their contests. On an average, 30+ external contests are hosted on our platform every month.

The URI online judge

In [25] is said that URI Online Judge is a project that is being developed by the Computer Science Department of URI University (Universidade Regional Integrada do Alto Uruguai

e das Missões). The main goal of the project is to provide programming practice and knowledge sharing. The web page entry words are: "Solve the available problems using C, C++, Java or Python, competing with other users. As a challengee, improve your ranking, solving as many problems as possible and tunning your source code to run fast."

First steps

To make a big start on programming testing environment first of all it was necessary to choose the programming language. Since one of the lines of objectives describes the use of an object-oriented language and unique studied in my course was Java, this question was closed. To make our system stable we could think about many solutions: first of all would be great to create such program that is independent from other systems like routers and its internet connection, a good example is cloud computing, since you put your software somewhere "outdoors" preventing to get a vulnerability of other unexpected source or manipulations. Imagine if you have the whole environment in your home, if there would be an electric discharge, your computer will shut down and the program will be unreachable.

In software cycle progress it is very important to have a kind of security testing as well as other tests, but none of the authors from bibliography explains in a good manner what and how should be done this tests. In book [17] it is only explained that this testing has the function to determine the access or data security of the software product and also it is done for security deficiencies. On the other hand, data security is a degree to which a product or system protects information and data so that persons or other products or again systems have the degree of data access is appropriate to their types and levels of authorization [26].

2.3 Technical workflow diagram

To design a diagram and to have some initial phase in programming and to see the skeleton and explaining so better the need of this system, we shall recall this from chapter 1, more concretely from section pre-alpha stage, the software design. Since we are constructing a program we can do the same analysis of requirements as was explained in software life cycle, and the part of tests the explanation will be done in next chapters.

The design diagram starts from the client-side approach and continues to the server-side interface, and after this explanation comes the programming language discussing.

2.3.1 Client-side interface

Figure 2.3 illustrates how client side work, but it must be mentioned that initially the client side was done as an installable GUI program, and for reasons of management was re-evaluated and was chosen the best option - a web interface instead of a program. Another problem while as a GUI program that was detected, was a limited client connection at the same time, as we would do a testing side also with sockets which converges to a limit of amount of users that are possible to connect at same time.

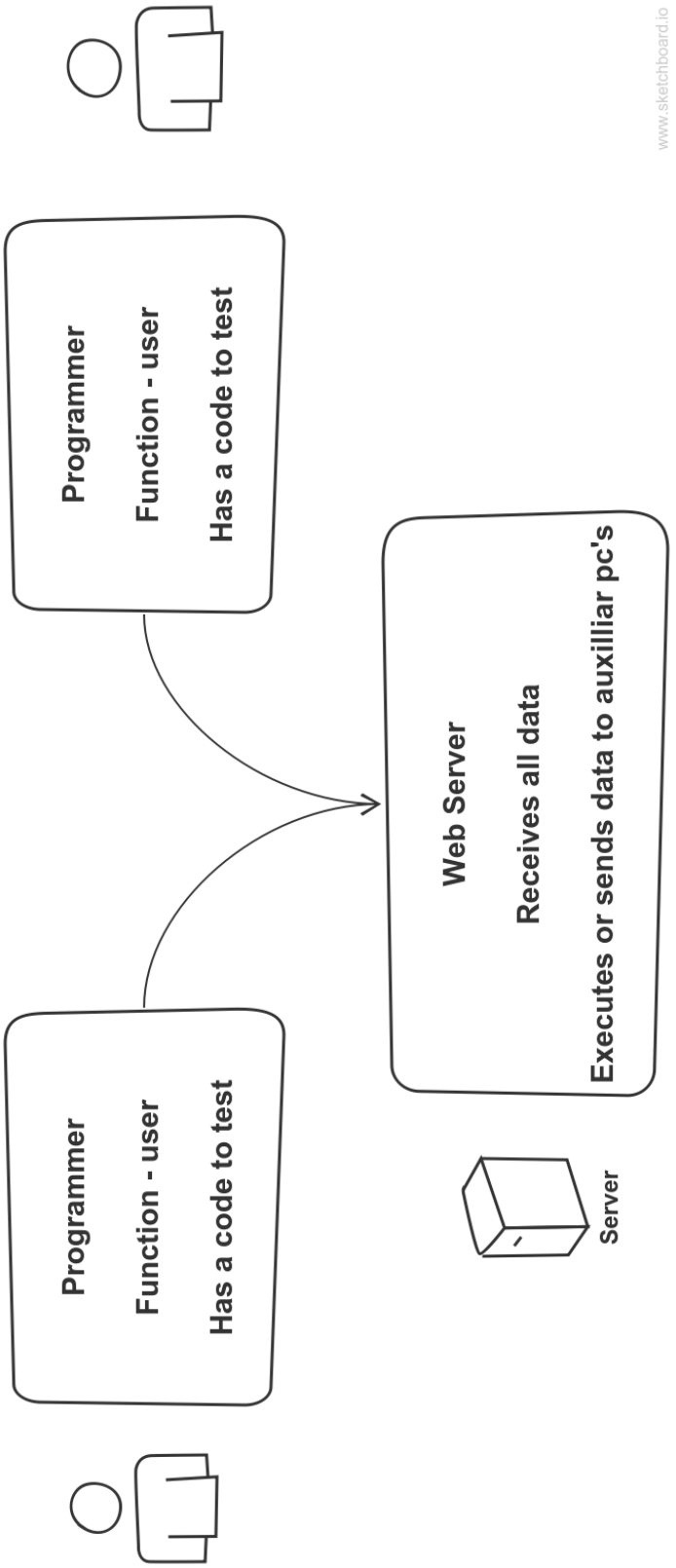


FIGURE 2.3: This client side design uses the same scheme as Mooshak judgement system, there is a programmer that will post his executable code for evaluation.

While having a connection established we should consider some kind of structure that will prevent users getting access to other users work, so it was necessary a login page with registration and login process. From figure 2.3 the programmer has a function of the user, with its "code" to test, in our case would be executable, which will come through a link made with the help of web server interface. This link can be used as well by others in the same way that uploads some data to any web server on the internet.

2.3.2 Server-side interface

To accept the data from programmers we should have a server that will have to make some tests. Since we are limited to the operating system or to the scheduling of threads that a multi-core processor is capable to launch, a great idea would be a multi-computer system. This could be acquired by use of more than one computers, and doing so we can process more than one programmer's test at a time with a good effort. Figure 2.4 above illustrates the continuation of figure 2.3.

Now comparing this system to Mooshak and as it was written in paper [18] about Mooshak, it is possible to configure more than one computers but each of one would have a website integrated into it and will be at same time testing server. Separating the testing process from web-GUI 2.4, where on one computer we can put a web server and other computers as testers will perform better than actually is done Mooshak. And this server could not be very powerful since its task will just redirect the files that someone wants to test, so it is possible the use of one raspberry pi (about what is raspberry pi [27]) that will be enough to have this all system workable.

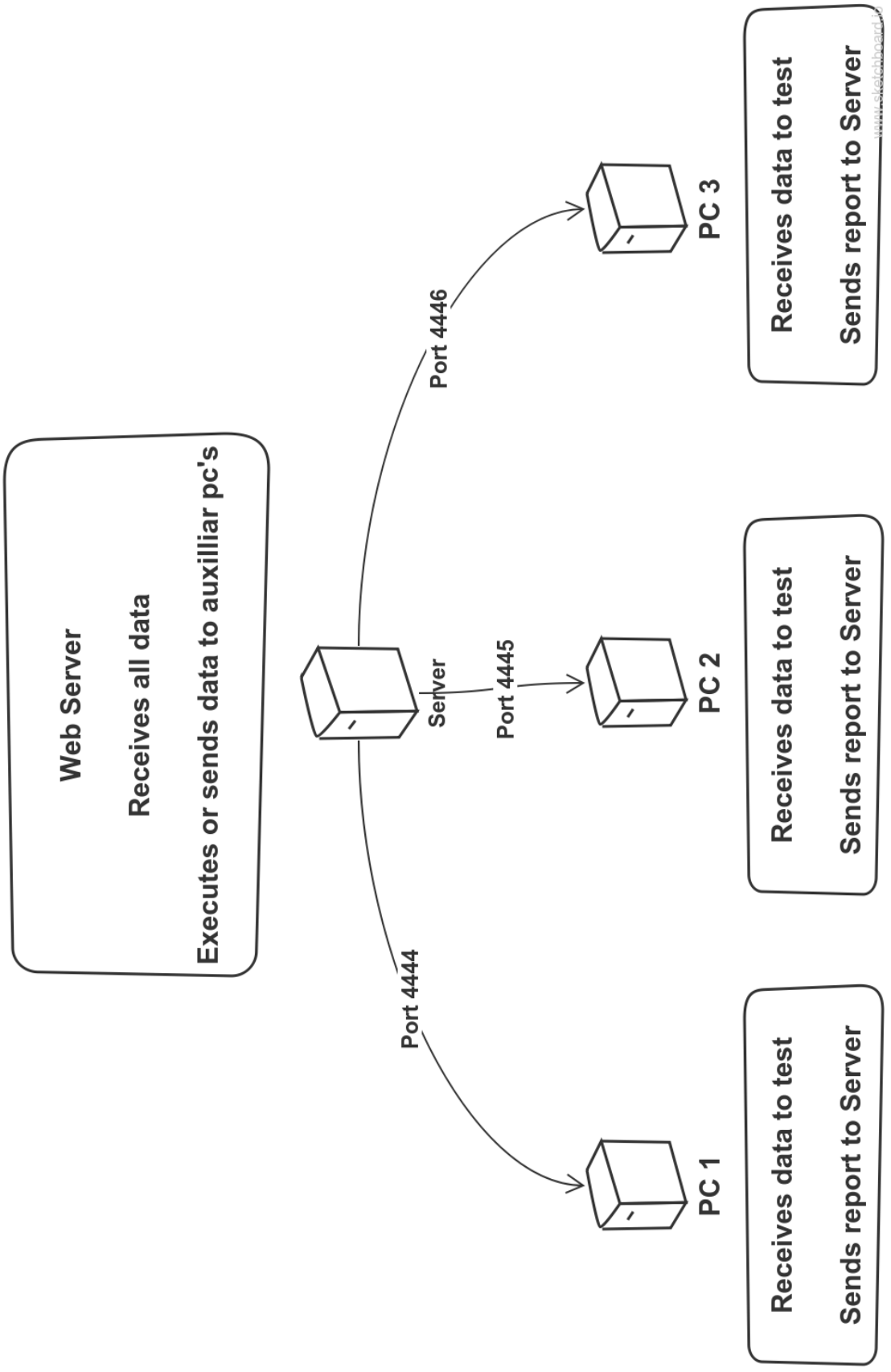


FIGURE 2.4: This server side design uses the same scheme as Mooshak judgement system, there is a programmer that will post his executable code for evaluation.

2.4 Programming languages

2.4.1 PHP , http, html

According to [28] and Computer network lessons from my course, HTTP is a communication standard that governs the requests and responses that take place between the browser running on the end user's computer and the web server. As we know server's job is to accept a request from the client and attempt to reply to it in a meaningful way, usually by serving up a requested web page. HTML is the standard markup language used to create web pages. Web browsers can read HTML files and render them into visible or audible web pages. But the problem came in constructing all possibilities of the testing environment was of what web programming language to choose. It took me a long time to consider PHP as one of them, but since I didn't have enough knowledge of any of others and at the same time finding one that would be better adapted to this work conducted me to the PHP. It looked me familiar with bash at first because in the declaration of any variable it is used $ sign (for more information about bash script you can read in [29]). But looking better at all procedures while constructing a website I made a conclusion that after all, it is not so similar.

While reading PHP [28] I started to think how I will join it to Java, and after some research I finally found a Java-PHP bridge libraries but couldn't understand how to use them to create a web server, and only after again more research I understood that this was accomplished by deployment to a tomcat server.

2.4.2 Tomcat server

From [30] Apache Tomcat is an open source software implementation of the Java Servlet, Java Server Pages, Java Expression Language and Java WebSocket technologies, which are developed under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2.

Tomcat also has a facility of importing war files that would favour us an easy portability of server package in our case the container. This import is done by the deployment of the project directly on the server. Tomcat has a relatively stable preservation, it's light weighted and flexible for future improvements.

War file is a web application archive that consists of a JAR file used to distribute a collection of JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static

web pages (HTML and related files) and other resources that together construct a web application. Since all java EE containers support War files this is a good advantage for future upgrades that could be done to a java EE application.

This became certainly clear for me that I don't need any PHP knowledge to develop my own web server since Java EE provides me with all necessary tools.

2.4.3 Why Java?

A Java program can be run on any operative system, as long as a Java Virtual Machine is available. There are thousands of libraries that have been written until nowadays, and this makes possible to simply import and access them, and also there is a possibility of downloading and installing more other libraries.

2.4.4 JSP

As was intended before the use of J2EE will be the stage of programming of web-server. The use of tomcat server will interact with it using JSP - Java Server Pages. As [31] says, the environment extends the need for basic Java support, as we have our testing environment functioning as a tester. Any computer running JSP need to have a container, which is a piece of software responsible for loading, executing and unloading this JSP. A server-side dynamic content solution, CGI (Common Gateway Interface), is the reason of why container is required. With CGI a server passes a client's request to an external program, this program executes, creates some content, and sends a response to the client. Notice that tomcat initially is configured to use CGI as default.

Servlet performance can be attributed directly to a Servlet container, where this container is the piece of software that manages the Servlet life cycle. Container software is also responsible for interacting with a Web server to process a request and pass it to a Servlet for a response. With containers in mind, the next figure 2.5 of Servlets is drawn using a container to represent the single process that creates, manages, and destroys threads running on a Web server.

The container also manages JSP in exactly same way as Servlets, because of the same technology used in both (in next chapter 3). The main reason of why Servlets are good for us it is because of each user that makes requests to the server it is used a thread management, this turns to be useful in processor scheduling, if we had in Mooshak our limits were in processing because of web server and testing environment is on the same computer which in this way limits amount of users that were logging in and users that test their programs, in this new system we can handle it as a part.

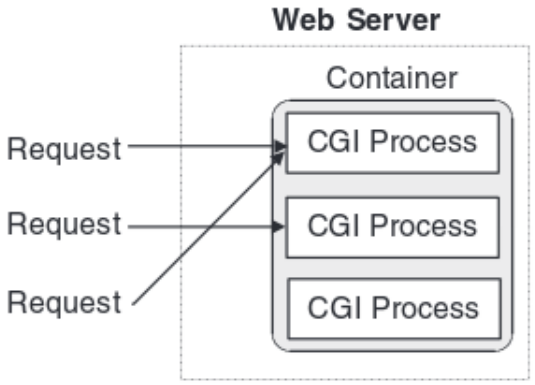


FIGURE 2.5: Container Processes as module in a Web-Server. Image taken from [31]

2.4.5 MVC form

According to [32], MVC - Model View Controller is a design pattern for the architecture of web applications. This concept is adopted to many languages, which purpose is to achieve a clean separation between 3 components of most any web application:

- Model: business logic & processing
- View: user interface
- Controller: navigation & input

In general, J2EE applications follow this MVC design pattern.

As it is shown on figure 2.6, the Model component - encapsulates application state, responds to state queries, exposes application functionality, notifies views of charge; View component - renders the models, requests updates from models, sends user gestures to controller, allows the controller to select view, and the Controller component - defines application behaviour in the manner to map user actions in future to model the updates.

Controller also selects the view for response and separates it one for each functionality. These components also interact with each other Controller for example, is the one who does invocations on both components, and controls the components, otherwise, the model component performs events on the viewer side and sends back the iterations.

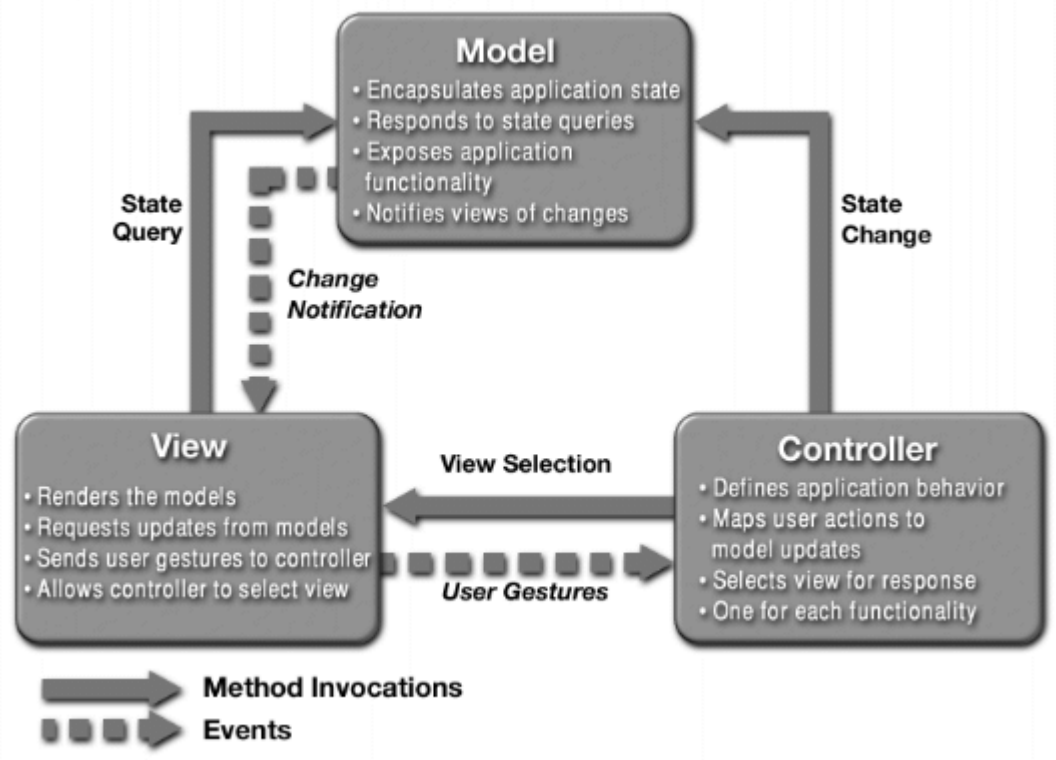


FIGURE 2.6: Generic MVC Structure. Image taken from [32].

2.4.6 Http session

Explanation of the implementation part of interface - HttpSession, is officially accessible in tomcat documentation from the next source [33]. It provides us with a way to identify a user across more than the one-page request, and this interface starts working after a successful login confirmation. The servlet container uses it to create a session between an HTTP client and an HTTP server, and then the session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit the site many times, and server can maintain a session in many ways e.g: cookies or rewriting URLs. The rewriting URLs method is used to send session parameter between JSP and servlet without encrypting data as an object, otherwise, cookies are not so secured because if the user deactivates it, this tracking is not applicable. This interface allows servlets to view and manipulate information about a session, such as the session identifier needed to separate different hosts or clients, the creation time and last accessed time of these clients. Also, there is a possibility of binding objects to it, allowing the user information to persist across multiple connections.

In this work, the HTTP session is used exactly for making possible the information to persist while interacting with the server. Imagine if this wouldn't be applicable, what

would happen next when 2 users make an HTTP connection and if both of them are trying to submit data for testing the generated report will be accessible only to the last user that will upload his data for the test. For this purpose, HTTP session is very important in our work, and as well because of multi-user utilisation and the necessity of tracking each of them.

2.4.7 Java Servlets.

As book [31] Java Servlets are an efficient and powerful solution for creating dynamic content for the Web Servers, and its power, behind Servlets, comes from the use of Java as a platform and as well as the interaction with a Servlet container as was explained previously in MVC form. A Servlet container provides us with - life cycle management, a single process to share and manage application-wide resources, and interaction with a Web server. To program with Servlets API it is necessary to have installed Java 2 Enterprise Edition (J2EE), and its official Servlet API is available for download on Sun Microsystems Web-page.

2.4.8 Master-slave concept

Analysing better the system more concretely the web server part, it could be considered that it is using a master-slave concept, where the master computer will be the web server and when an executable is sent for certain tests, slave's computer or auxiliary computers are waiting for a connection, this connection is made with the help of an executable that will be downloaded and launched. After it will be generated a report with all failures and this report will be sent to the web server which will present it to the user on the web page. During the upload phase, the communication type will be unidirectional. If the first slave computer is busy it passes to next one until it gets one that is ready for use, if all of them are busy it means that all socket ports are used so the cycle starts again looking for a port that is ready for use, acquiring so a new computer that is not busy. It must be mentioned that any of users can accept to use their computers as testers making it available to our environment network.

Chapter 3

Experimental setup

3.1 Web server

As we saw in the previous chapter 2 we will separate this programming part in 2 phases - the web server part and the computer dependency part. The web server has a login page and registration page as well as all initial data like identification parameters - username and password, which will be saved at server side in "login.txt". After a welcome page explaining what is this website used for, the user will be redirected to the login page where all data will be confirmed from login.txt. The error page will be presented if the wrong email address is inserted instead.

The registration page will contain important lines that will be signed by the asterisk and other lines will be used for future reworks. All this - login, registration and error pages are written as JSP and will send necessary data to the servlet, where it is processed like values of login information.

This communication process consists of Http session tracking and it is done to not to mix data of different users. From the moment that Http session is initiated it is used the session object as communication between JSP and Servlet. The control of this duplex communication is managed as MVC scheme. To make easier the upload the full process, when user wants to publish his work for tests, it was implemented unzip manager which creates a temporary directory of each user, and this directory will be deleted after all tests are done and it is saved a log of what kind of tests were made and which one were passed.

3.2 Supplemental computer

On the testing computer side we will have a simple socket listener where it will wait for a code to test, and as it is mentioned in objectives part in chapter 1 it will be applied BSD sockets. This means that sockets used were first done for Unix based systems, but this doesn't mean that it won't work on other platforms, this sockets were programmed for all operating systems.

3.2.1 Communication through sockets

One of the specifications present in objectives is the use of sockets for the communication between end systems, as it is known from Computer Networking concept this communication is done in transport layer, and there are 2 possibilities - the use of TCP or the use of UDP protocols, since we want a reliable data transfer it is implemented a TCP protocol. First of all, we should understand that we have to work with some data transfer mainly to the web server and after, it will pass data to the secondary computer that will do all the work and will send a report back to the web-server. This second computer thus in the first phase will act as a server, that will be waiting for the connections from the client and after establishing this connection the test suit will begin. In java socket, the communication is done bidirectionally and the need of the start of another socket is not required and as well as the port number will remain the same. Using threads we can adjust more tests and this tests would be done simultaneously. This is actually good because if we append more than 1 computer we can have more acceptance by users, this makes our project only dependable on connections that our web-server can have. Another thing that must be mentioned is that users would not upload their source code but only the executable, making us win time during processing of the program, and after this pass directly to testing part.

3.3 Coding phase.

In this section will be explained the base architecture of how it is made all execution of web-server.

Since Servlet is used as an interaction between a client and server, and understanding HTTP protocol it is possible to add that these similarities can be rewritten and recalled as Http Servlets. Next, it would be made a brief introduction of 2 methods of HTTP but consulting [34] you can provide yourself with better knowledge of HTTP as a protocol. Going back to Http Servlets we shall explain the 3 parts of its life cycle:

- Initialization
- Service
- Destruction

They are basic functionalities of a Servlet container that is using Java to dynamically generate the web page on the server side. A servlet container is essentially a part of a web server that interacts with this servlets.

The initialization and destruction methods may be performed once a time whereas service can be done many times. Initialization is the real load of resources, during this process all necessary parts are initialized (all Servlets must implement the javax.servlet.Servlet interface). When a container loads a Servlet, it invokes the init() method before servicing any requests. Services will accept all requests from other outer-systems(users or other secondary servers).

The service() method is invoked once per a request and is responsible for generating the response to that request, and also the Servlet specification defines the service() method to take two parameters: a javax.servlet.ServletRequest and a javax.servlet.ServletResponse object. By default a Servlet is multi-threaded, meaning that typically only one instance of a Servlet is loaded by a JSP container at any given time. Finally, the destruction phase is initiated when a Servlet is being removed from use by a container. Each time a Servlet is about to be removed from use, a container calls the destroy() method.

3.3.1 GET and POST

Since the HTTP is used to make reference to all internet worldwide pages in servlets there are 2 most important methods that are used they are: the Get and the Post (figure 3.1)

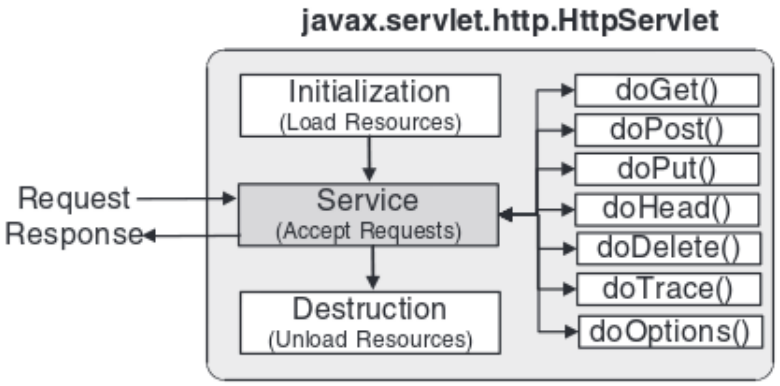


FIGURE 3.1: Http servlet scheme. Figure obtain from [31]

These two parameters appear as the consequence of service part of the schematics, however, there are many other methods that service() "pack" possesses. The get method is great for sending a small amount of information but it will appear appended to URL and this is considered unsafe, for example with the use of Wireshark program, which is literally a network packet sniffer, it is possible to see the information that was asked or given from server to the client during this Get method execution, but anyway it is normally used while executing queries in some Web application, when you want to catch some resources like images you use this kind of methods, otherwise the Post method is used.

The difference of 2 methods is that Get method requests data from a specified resource while post usually submits data to be processed to a specified resource.

The post method has similarities with get method but at same time it uses a bit different mechanism on sending data, get for example, like was said previously, is used for sending queries or any kind of URI, so some browsers make limitations in terms of characters e.g. Internet Explorer limits it to 2083 but this can not only be limited at browser side it is also limited by server.

The post method has unlimited information flow and can use queries as requests like get method does, also the information that is sent from both ends is done with a socket connection. The data flow is encrypted and hidden from others during this process and

basically if a user wants to update the data from HTTP request that is sent from the query and this data is bigger than 255 bytes, then this "transfer" should be done with the help of the Post method, but there are also other methods available, as it is shown on figure 3.1, where each of them could be called once per request, occasionally calling each of the methods has a consequence in response code and this all is afterwards processed as exceptions in case the method returns a negative response. To write a Servlet we should start with the next lines:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
   public void doGet ( HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException
    \*** ... some code here ... ***\
    }
```

Listing 3.1: doGet

The doGet method can be changed with doPost because both of them have same arguments and are treated as was explained before. Deployment of the servlet is done with help of web.xml file located in /WEB-INF/web.xml. When all this code is written, launching the web.xml will call the main class that would be written in web.xml as a servlet-name parameter. There is also a possibility of deploying the server program to some tomcat server, all that is needed is to export the junction of classes that make the server work as a .jar file, and going to tomcat server page (http:127.0.0.1 or http:localhost) and deploy it there. To access the web page from another computer you should insert the IP address of the computer where the tomcat was installed, this can be done being on the same network as a server and to access it from outside of the network it can be done redirecting the router port to the tomcat port 8080. Keeping in mind a 12 digit IP address can be a difficult challenge so there is a possibility to pay to a web hosting company that will attach to your IP address a name of some web-site of your choice.

Returning back to our objective the Get and Post methods are important in this stage since this will make the complete conversation with a servlet. Analysing Servlet example code, we should mention that there are 2 objects that are present as arguments in doGet function HttpServletRequest request HttpServletResponse response which treats request and response respectively.

3.3.1.1 HttpServletResponse object

This object is most important because it is the way the server "talks" or sends data to the client, producing so an empty Http response. To send back the custom content there exist a requirement in the use of getWriter() or getOutputStream() methods that make part of HttpServletResponse object. Calling this method we can write something that in the future will be sent to client from the string to any binary data that can have some encoding type. To attempt the use of both of this methods there is a requirement of the implementation of exception sentence, more specifically when it is used the getWriter() function in the construction of an HTML text, there must be used a certain syntax and in the case of not following of this syntax the exception occurs. The example of the book [31] writes a - "Hello World!" on the web page.

```
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Hello World!</title>");
 out.println("</head>");
 out.println("</html>");
```

Listing 3.2: getWriter() method example

In other hand getOutputStream(), as was mentioned previously, is used for sending a raw byte data, this means that it is possible to send any kind of data. For our purpose as software testing environment, we will test an executable file so the use of this function should be reasonable. While entering more and more in details of the properties it is known from computer networking [35] that HTTP-response is composed of the next elements: status line, header lines, a separating blank line (Carriage Return + line feed) and an entity body (the figure above represents the exact "syntax"). To manipulate all these elements HttpServlet object contains the next methods:

- setStatus() - sets a status line (example: status code 200)
- addHeader(String name, String value) adds a response header with the given name and value. This method allows response headers to have multiple values.
- containsHeader(java.lang.String name) returns a boolean indicating whether the named response header has already been set.
- setHeader(String name, String value) sets a response header with the given name and value. If the header had already been set, the new value overwrites the previous

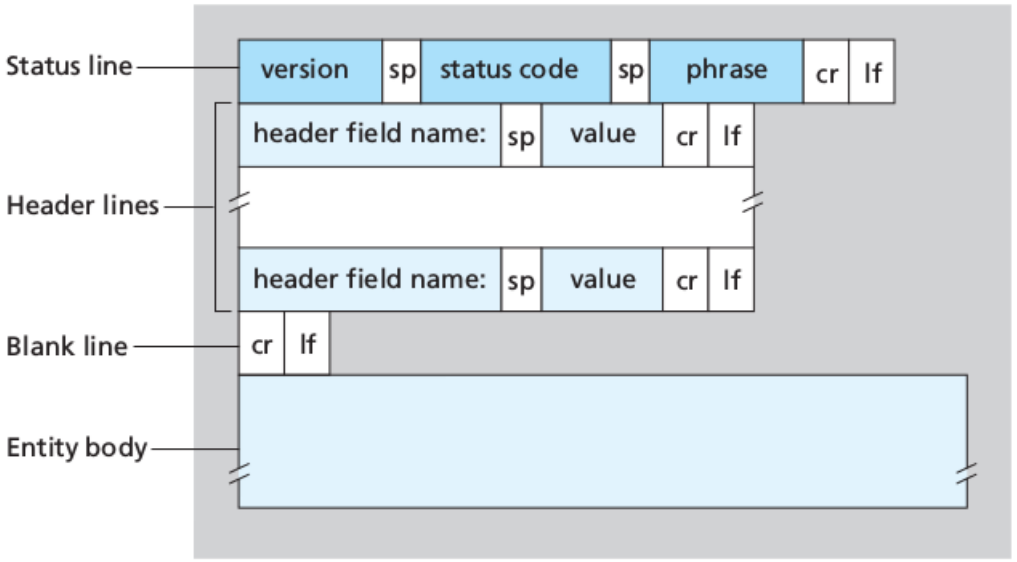


FIGURE 3.2: Http response format.(from client view)

one. The containsHeader() method can be used to test for the presence of a header before setting its value.

- setIntHeader(String name, int value) same as previous but with difference that the value now is integer.
- setDateHeader(java.lang.String name, long date) sets a response header with the given name and date value.

Http request header methods can help us in identification of all sorts of information like the time on which it was created, what is the operative system that server is running, what is the language that the client is using, the data to be presented (not all characters use the same encoding and should be translated to its "visible" code), what is the web-browser is it used on client's side and many others.

3.3.1.2 HttpServletRequest object

HttpServletRequest object is used for getting request headers, when we first insert the web-site in the browser an HTTP request is constructed with all normal information described previously like what is the web browser, IP address, time of creation and etc., and all this data is sent to the server. Treating that information as necessary we have some methods almost same as response methods, previously seen, but with the difference instead of "add" or "set", we have the "get" part.

- getHeader() - method returns the value of the specified request header as a string.
- getHeaders() - method returns all the values of the specified request header as an Enumeration of String objects.
- getHeaderNames() - method returns an Enumeration of all the names of headers sent by a request. In combination with the getHeader() and getHeaders() methods, getHeaderNames() can be used to retrieve names and values of all the headers sent with a request.
- getIntHeader() - the difference from the getHeader it is just while the header has an integer value it passes it directly into int saving so programmer don't need to use java methods like parse to int the string that would be returned in getHeader().
- getDateHeader() - method returns the value of the specified request header as a long value that represents a Date object. The date is returned as the number of milliseconds since the epoch. The header name is case insensitive. If the request did not have a header of the specified name, this method returns –1.

The use of all these methods/functions can help us to determine the operative system that is used by the user and it can help us preventing an argument which contains date header part and all this must be done in such way that there must not exist any mistake from another header type. The operating system header is fundamental to us to know what type of executable we should test.

For this case we don't need a direct interaction with the user, but what if we need one? How would we ask a user to upload his program? We already understood that there are two main objects created for HTTP communication, then one of this objects should have a method for "encapsulating" not only headers but also other parameters.

- getParameter(String) - returns a string object representing the corresponding for the String value.
- getParameters(String) - same as previous, but used for mutliple parameters with same name.
- getParameterNames() - this method returns a java.util.Enumeration of all the parameter names used in a request. In combination with the getParameter() and getParameters() method, it can be used to get a list of names and values of all the parameters included with a request.

Using methods described upper, we can construct a complete interaction with parameters that client have passed to us, for example when the servlet is programmed to make a login confirmation one of the parameters would be the login and another - password, since this is done there must be a file or a database with all login and password combinations and getParameters() will find both arguments passed by client and a simple function for accessing a file with all logins and passwords will make the confirmation. The difficult part comes if the user wants to upload his data to the server like a picture or a .zip, it would be necessary to make the parting of this data and also the data type should be known by the server. To know the data type there is a function getMimeType(), this function returns a String with type of content used primarily from HTML web-site, but of course we can't use the getParameter() here because it only gets parameters of ASCII encoding and translates it to Servlet as a String. To make possible the upload we shall add to HTML <form> the enctype argument (HTML programmers can understand why and where to do this) and pass it as multipart/form-data, then from servlet making use of getInputStream() will make initiation of download of partitioned object and will join it one by one (from server side is download while from user is an upload), the reason why this happens is because the data while travelling on the internet is limited by maximum transfer unit that is limited by server.

To send different requests between many servlets it is necessary to mention the next methods:

- getRequestDispatcher(String path) - The getRequestDispatcher() method returns the RequestDispatcher object for a given path. The path value may lead to any resource in the Web Application and must start from the base directory, "/".
- forward(javax.servlet.ServletRequest, javax.servlet.ServletResponse): The forward() method delegates a request and response to the resource of the RequestDispatcher object. A call to the forward() method may be used only if no content has been

previously sent to a client. No further data can be sent to the client after the forward has completed.

- include(javax.servlet.ServletRequest, javax.servlet.ServletResponse): The include() method works similarly to forward() but has some restrictions. A Servlet can include() any number of resources that can each generate responses, but the resource is not allowed to set headers or commit a response.

Since it was explained before in the MVC form, would be present a need in passing request objects between servlets, with the main objective of controlling and manipulating data or other parameters. The server side explanation ends here, it doesn't mean that all processes were explained but the most of the essential part - was, the next stage consists in explaining the use of JSP(Java server pages).

3.3.2 Java server pages

Java Server Pages and Servlets are both necessary to produce a separation between view and controller while coding of a web server in java. This separation is done by dividing web pages format text from the servlet and an integration is done with the help of external "online" functions. This means that every time when a JSP is revised and needs to be recompiled the necessity of reinitialization of a server is not required, this facility helps us to update the information without loss of connection from the server.

In the first part we talked about communication between user and server and recalling this, it is done with HTTP requests, and further, we introduce the HTML tagging reference, so it is the time to introduce that it is here that we can design a HTML web page.

The result of JSP acceptance, all web marking language like HTML, CSS and it also makes possible use of javascript and PHP tags, and as well the major point of why to use JSP is that its mechanism makes a link what would normally be static markup to custom Java code, this exchange helps a java programmer to adjust his java program to every kind user-server interaction. For all these statements it is the most fundamental reasons as me as a programmer to chose Java server pages and servlet libraries to make all the interaction being possible without any lack in the programming language.

3.3.2.1 JSP life cycle

Java Server Pages(JSP) is designed especially to simplify the task of creating text producing HttpServlet object and it is done by eliminating all the redundant parts while coding a Servlet. Unlike with Servlets, there is no distinction between a normal JSP and one meant for use with Html. All JSP are designed to be used with HTTP and to generate dynamic content for the World Wide Web.

If we look at figure 3.1 again we can see that servlet has its life cycle divided in 3 parts so as same as JSP does:

- initialization
- service
- destruction

both JSP and servlet contain the same life phases, the difference between them is in the methods that they use, where initialization is called by jspInit(), the service jspService() and destruction method jspDestroy(). Some of this methods are initiated with the server and other like destroy are executed as final steps with intention of shutting down the processes.

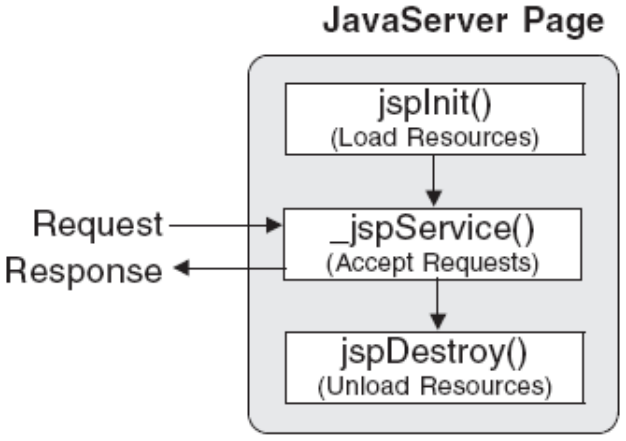


FIGURE 3.3: JSP life cycle.Image obtained from [31]

During "jspService()" process it is made the loading of the view design of the web page. The jspService()method corresponds to the body of the JSP page, and it is defined automatically by the JSP container and should never be defined by the JSP page author.

Next is presented a small program that will present "Hello World!" string in bold, with the title "Hello World" and its syntax is written in HTML.

```
<html>
 <head>
 <title>Hello World!</title>
 </head>
 <body>
 <h1>Hello World!</h1>
 </body>
 </html>
```

Listing 3.3: JSP example

This example shows only the use of HTML tags that are different from servlet tags because in servlet example we should have this HTML code tagged into the HttpResponse object, here we don't need this "encapsulation". Thus the JSP helps us with simplification in web development part and the use of MVC form is essential.

Coming back to JSP coding, it is possible to make use of java syntax inside as well, this is achieved with so called "scriplets" and it is done with this tags - "$<\%\ \%>$", putting any java code between this tags will work here. For example to use the "for" cycle we should write the next code $<\%\ for(inti = 0; i < 10; i++)\%>$. In practice, it will be simpler to understand the scriplet code if you have an experience in java programming language.

As it was mentioned before the JSP life cycle (figure 3.3) consists of 3 parts, and if we program a scriplet we need to know where we will declare our variables, because if we start our marking tags for scriplet before the HTML code, this code will "happen" before jspService(), if we put it after jspService() the code will happen before the destruction but after the HTML, and this makes our code resist for 1 session. Putting a scriplet before JSP we can make a simple counter of how many times the website was accessed. To make the possibility of it being multi-sessional we should pass the variable from servlet making an external variable as a counter. To make an interaction with servlet the HttpServlet object.forward() method that was seen before will work here, as other methods and therefore exists a possibility to add other libraries for example $<\%@pageimport = "java.util.List"\%>$. Next, are shown other tags possible in JSP:

- $<\%--comment--\%>$ - Documents the JSP file, but is not included in the response.
- $<\%!declaration;[declaration;]+...\%>$ - Declares variables or methods valid in the page's scripting language.

- $< \% = expression \% >$ - Contains an expression valid in the page's scripting language.
- $< \%codefragment \% >$ - Contains a code fragment valid in the page's scripting language.
- $< \%@includefile = "relativeURL" \% >$ - Includes a file, parsing the file's JSP elements.
- $< \%@page \% >$ - Defines attributes that apply to a JSP page.
- $< \%@tagliburi = "URI" prefix = "tagPrefix" \% >$ - Defines a tag library and prefix for custom tags used in the JSP page.
- $< tagPrefix : nameattribute = "value" + ... > \|$
  $< tagPrefix : nameattribute = "value" + ... >$
  other tags and data
  $< /tagPrefix : name >$ - Accesses a custom tag's functionality.
- $< jsp : forwardpage = "\{relativeURL \| < \% = expression \% >\}"$
  $\{/ > \| > [< jsp : paramname = "parameterName"\ value = "\{parameterValue \| <$
  $\% = expression \% >\}" / >] + < /jsp : forward >\}$ - Forwards a request to a web resource.
- $< jsp : includepage = "\{relativeURL \| < \% = expression \% >\}"$
  $flush = "true \| false"$
  $\{/ > \| >< jsp : paramname = "parameterName"$
  $value = "\{parameterValue \| < \% = expression \% >\}" > +$
  $< /jsp : include >\}$ - Includes a static file or the result from another web component.

As you can see JSP has a vast variety of tags and it is possible to make everything that you could do as coding in a java program.

Chapter 4

Software testing

In this chapter, it will be presented how this software testing environment operates. Since we already made a brief introduction in chapter 2 about how will look the client side and the server side, we should look as well for examples of other testing environments that we talked previously (lets chose Mooshak), it is possible to understand that putting the web-server in the same machine where the testing environment is, will be a bit risky. This is because if the computer where all environment is launched fails the possibility of testing, user's code will no longer be available. A solution can be distributing the test environment by more than one PC, so if one fails the testing, the process continues in others.

This software testing environment workload distribution is implemented in 2 different ways, as proposed in [36], for a cloud based on a grid computing architecture over the Globus middleware. However the implementation presented here has no middleware support. With the first strategy the submitted test sets are executed on a server computer. To distribute work loads the test sets can be distributed on a set of slave computers, according to a master slave architecture. These are similar to Mooshak workload distribution [18]. The second strategy is also based on distributing the test sets over a set of computers, however these computers are the very same user computers that submitted the test, thus reducing the hardware required.

4.1 Implementation of the Software testing environment

4.1.1 Work flow

First of all the web server can be any computer with tomcat with version 7 or above installed, and it can run even on a very low resource computer like raspberry pi, while as a testing machine that can be called as "plug and play" part, must have a Java virtual machine installed, with objective of being part of this "schema". This is enough because the environment does not compile any source code, it will only run an executable that should be uploaded in the process, for example in Linux it's a .elf or .sh, in windows is a .exe and in mac an .app. If the user wants to "donate" some of his processing resources to help this environment, there is a binary version of the testing machine that can be downloaded by him, as a .jar that if executed will automatically make a direct connection to the main web server. This way the server sends the test cases to the user's own computer, where the actual testing takes place, being only monitored by the environment. The user after all receives a .jar as executable which is a part of the server and .dat with input/output that can only be read by the .jar file. This way a .jar file consists from socket connection and case which user have chosen.

This real scheme figure 4.1 consists in the better understanding of how it can work in perfect flow.

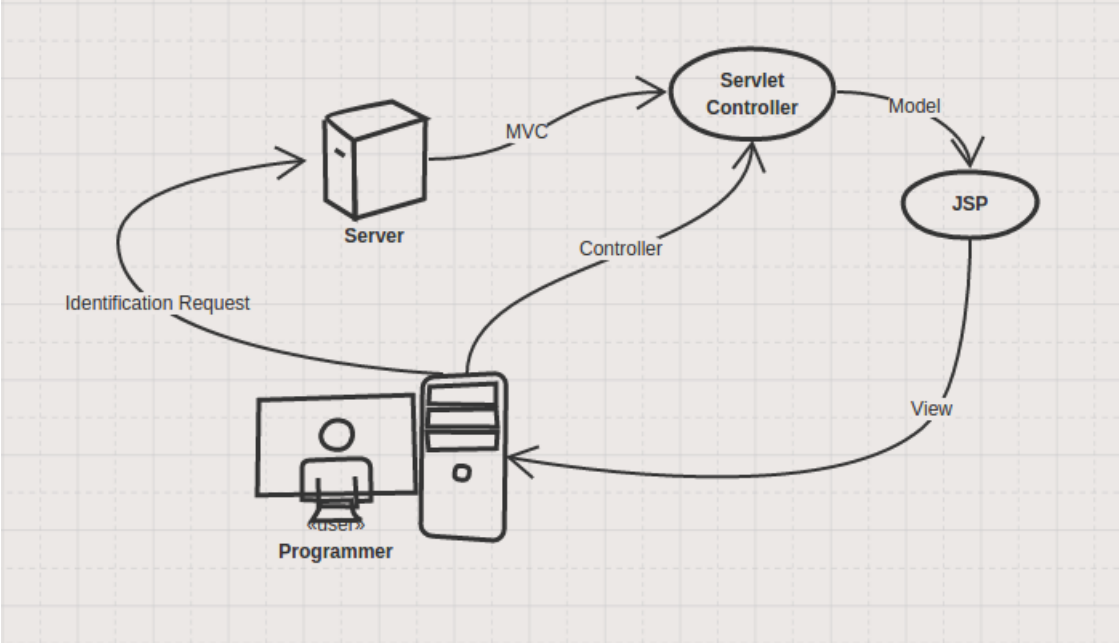


FIGURE 4.1: MVC interaction.

For the first step, the user enters into the web-site and he has 2 choices: to create an account if he is a new user, or to insert login information and password. From the figure 4.1 we can see the design of the MVC form described in the previous chapter, and there would be a session initiated after (and never before) the login confirmation. It is done this way to secure that every time that some new user enters will not cause an extra processing that will be dedicated for this "link".

After login is done the next step consists in uploading the executable file, so, here we have 2 possibilities the first one is to test if the executable does provide the output that is required, then imagine if we have 10 test cases, testing the first case and if it passes - we go to the next test case, if the error occurs, a message is sent to user with that test case error. Note that it is never presented all test cases at the same time, only that one that the user fails. If suddenly the connection will go down, when it is used users contribution the socket is closed and the process will have to be initiated from the choosing the test case part. When it is used our environments computer, it is implemented a session process where each user has 1 hour of inactivity, after that time re-logging again will resolve the problem and the last seen page will appear again. To know if it is a real program and not a copy of the solution of each test case it is possible to limit a minimum size of this executable file, controlling the size will make us know if the file will change after the execution. In the case of some malicious software, normally what happens is that it tries to access the process that is launched and adjust some new properties in that executable that is being analysed in this manner that the size of this program also is modified. This kind of security will prevent future inconveniences like system reboot, that would corrupt the work of testing environment in overall.

4.1.2 User contribution

Since the environment is programmed as 2 separate parts where there are different interactions between user and server as itself, we can also ask about the use of programmers computer. This contribution will benefit us in a very small system, imagine that all programmers that want to test their programs will accept this use, there would be a link created also at web server side as JSP page where the programmer can download a testing unit of environment and after executing it, the web server will recognise its computer and will save some good processing time. If this will be accepted by all users the necessity of only a web server will be enough to make the complete environment.

As it was said before the web server system don't need a strong CPU and any device with Linux operative system (just for example) installed there with tomcat and java (both updated at least to 7.0 and 1.7 respectively) will be enough to make this all work reliable. The next figure 4.2 shows how it can look like.

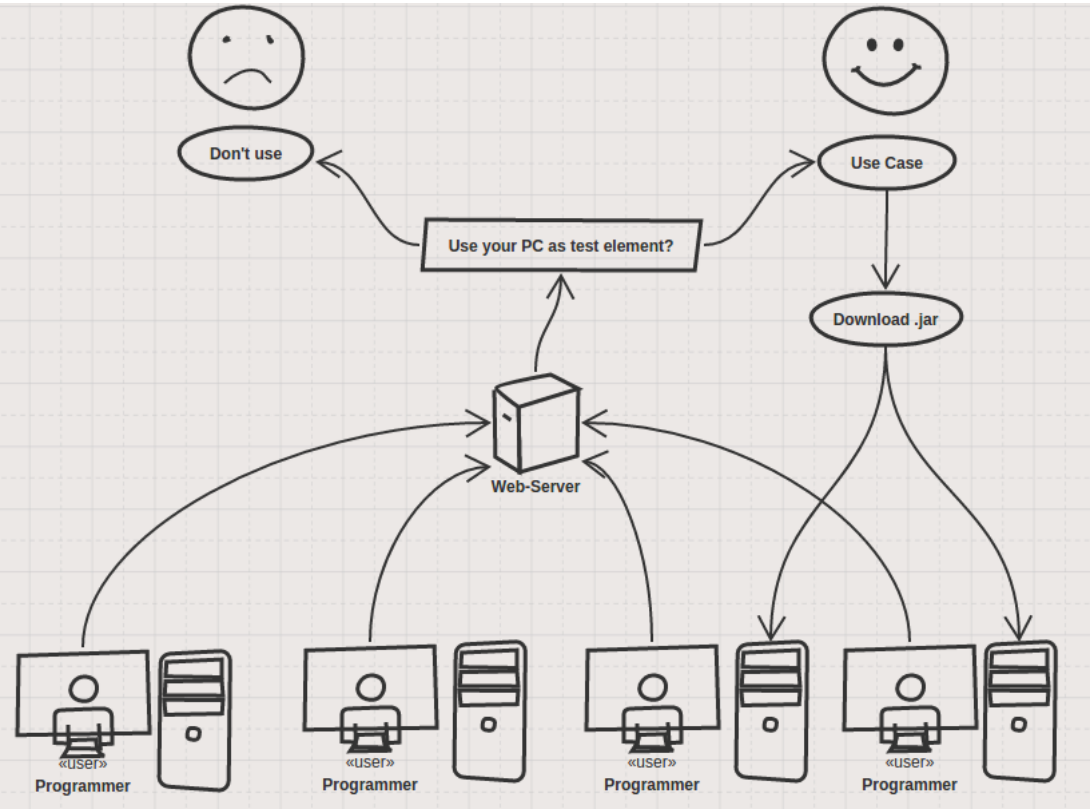


FIGURE 4.2: Testing unit.

Looking at the figure we can see that 2 of 4 programmers accepted to be contributors, the question you may ask is if their computer will be used by other programmers that didn't accept the contribution, so in the contribute page will appear a check box asking if they do want or don't want to be contributors. The use of the session is made after

the login page and there would be added a variable if the user is the contributor and also the level of contribution for example level 1, and level 2, where level 1 is for making able the use of his computer for others and level 2 for only him. The limit of the session is not decided yet, but if the computer that serves us as test element will go down or will be disconnected from the internet the backstage computer will become the main testing unit as initially was pretended to be done.

4.1.3 Server as itself

At this stage, it is possible to make a simple and cheap software testing environment system using this "concept".

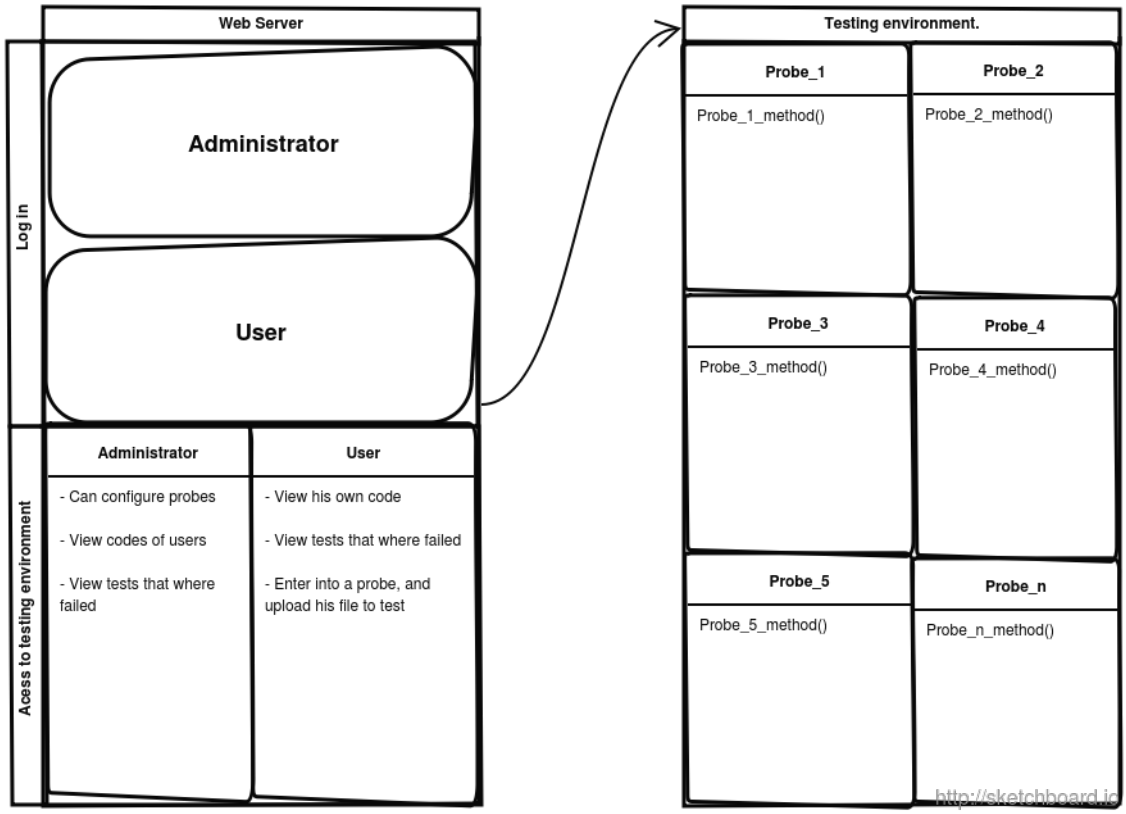


FIGURE 4.3: Software Testing scheme

The scheme from figure 4.3 separates the environment in two parts where the first one already explained - web server and the second - testing environment. The second part consists of probes that user will want to choose as this work will not be done just for one kind of probe. The choice of probe will depend on from what user want to test, if the user will accept to use his computer as tester, it will help us in the manner that there would be a necessity on installing some libraries that don't exist on our computer, since user wants to test this kind of software he should have it installed on his computer and

if the user provides his collaboration with the problem will be resolved. Looking again at the figure 4.3 we have the web server divided into 2 phases the login phase and access phase where the login, we already saw, will help us to know if does there exist this user, but also there would be an administrator for users. This administrator have a whole control of the system from the scratch, it means that it is administrators job to program probes and provide them to the user as viewable parameter, also administrator will have the privileges of watching who is testing and what, this is not a direct access, but at the moment that user uploads a file it is created a file on web server, that consists of the user name of who did the upload and the date of creation of this file.

The testing environment part as was said before will consist from probes, the user that will want to contribute with his computer will download the probe, and this probe will be executed after the confirmation from web server, this doesn't mean that contributor would not upload his file to the server, actually the web server is who will perform the test, but on targets computer indirectly.

4.2 Before the execution of the Software testing environment

It is expected from user(programmer) the next knowledge before execution of tests on this environment:

- first of all - the objective of the application should be well understood by the programmer (this objective can be provided in probe page).
- after the application construction, the simple run test should be performed by the programmer as well, just to not make use of unnecessary process on the server side if the program is not working.
- after uploading of the program some tests are performed, which supposedly programmer doesn't know.
- programmer gets a report according if all tests were passed or not, until the last he failed are demonstrated.
- there will also be a log page where the basic data is written, but it will only be accessible by the designer of the tests.

All these steps must be performed for concrete accomplish of the objective, this is very meaningful in terms of the success of the correction of the program that is being tested.

In the next chapter will be discussed the way how probes work and as the example will be provided with a certain and concrete test case where a student or programmer will upload his program and will pass through the suit of tests. Also as it was demonstrated previously the chart how the work is structured, here we will enter more deeply to understand better this part.

Chapter 5

Case Studies

While executing the case studies it should be mentioned that in the master node it was developed a web site for uploading executable on which will be made tests that where defined previously by the creator of this tests.

Over all, for each client PC will be available testing environment prepared previously by the project manager. This environment will consist of a software testing part which will apply a testing case scenario. This scenario can consist of an input/output cases, which previously only project manager knows.

There are two possible forms how we can do this tests, with or without user contribution. The difference between two forms are that the user contribution .jar consists only from one test case that was chosen previously by himself, while without contribution, system is prepared with all auxiliary computers that integrate all possible test cases.

Test cases are API objects - probes, that can be constantly in developing stage, because of full possibilities of what kind of problems could appear to accomplish the objective. These API objects after finished could be recorded in object files that are sent to a client PC, which can be done through a socket. On the figure 5.1 has demonstrated a diagram of this API.

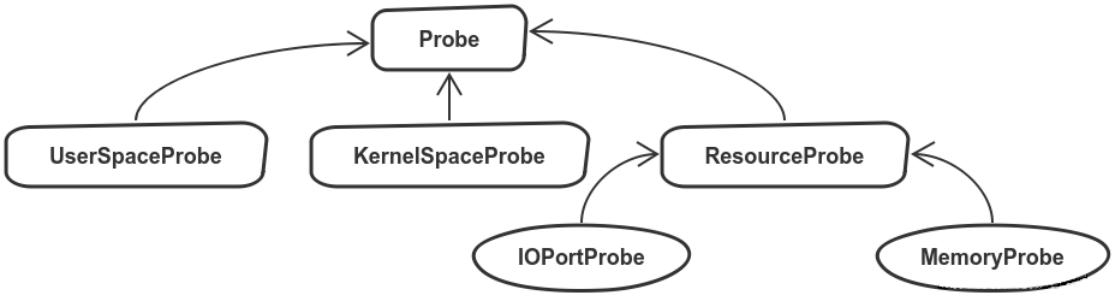


FIGURE 5.1: Probe hierachy (UML)

The objective of this work is not to implement this API as itself, but at least to make a complete testing environment that will distribute and execute this objects over the case tests.

As an example of creating an object of simple probe class hierarchy (a test case), will be a probe in the user space. In this case, a simple set of the entry is applied where the expected output after input of 2 Strings "2 2" is "4":

StdInOutProbe p = new StdInOutProbe ("2 2", "4")

after the implementation of object p, this object is saved as a file (.dat) and it is sent to the testing PC. On the testing PC this file will be found and read, then the test is executed and expected output is compared to real one provided in the construction of testing probe.

This is a brief introduction of what and how this probe API is done, but next, it will be provided with the more complete explanation of each step.

5.1 Case study 1

In this case study will be described how to add a new probe class, but first of all the user should analyse the code and see if this probe class already exists. There are already 2 probe classes that are ready to use in the library which is - network probe and userspace probe.

First of all, we shall start by describing the fundamental part how it is implemented and what is necessary to implement or inherit before starting to write a new probe. This is described in ProbeAPI subsection, and the next subsections will explain how this 2 probe classes are working.

5.1.1 ProbeAPI

In the ProbeAPI hierarchy there exist 3 classes to facilitate the construction of new one, they are essential for making the next step for testing.

All test probes must have as base class TestProbe and implement Probe interface methods. If they are classes that descend from TestProbe which already implements Probe by inheritance then these classes implement also the Probe class, which is the interface for probeAPI hierarchy.

Stating from the next page it is added the code of the probeAPI hierarchy.

5.1.1.1 Probe.

```
package probeAPI.probes;

import java.io.IOException;
import java.io.OutputStream;

/* All test probes must have as base class TestProbe
 * and implement Probe interface methods
 * If classes descend from TestProbe which already implements Probe
 * by inheritance these classes implements also Probe
 */
public interface Probe {
   public void testOK();
   public void testError();
   public void test();
   public boolean compare();
   public boolean report(OutputStream os) throws IOException;
}
```

Listing 5.1: Probe

5.1.1.2 TestProbe

```
package probeAPI.probes;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Serializable;

//Must implement serializable so that sub-classes can be stored to file
abstract public class TestProbe implements Serializable/*, Probe*/ {
   private static final long serialVersionUID = 1L;
   protected static final String TESTFAILLED="FAILED!";
   protected static final String TESTPASSED ="PASSED!";
   protected static final String TESTERROR ="ERROR!"; //System error
    performing test
   protected boolean testDone=false;  //false before test accomplished
   protected boolean testError=true;  //true before test accomplished, false
    if no system error (Exception) when performing test
```

```
   protected String  testErrorMessage=""; //test errorMessage of Exception
    occurred when performing test
   protected TestCase testcase;

   public TestProbe(TestCase testcase) { this.testcase = testcase;   }
   public boolean testDone() { return testDone; };

   //Concrete classes must call one of this to signal if test performed OK
   //or any system error occurred
   public void testOK() { testDone=true; testError=false; }
   public void testError(Exception e) { testDone=true; testError=true;
    testErrorMessage=e.toString(); }

   //to be defined in descendants each class can have
   //different methods of testing and comparing
   public abstract void test();
   public abstract boolean compare();
   public abstract String testDescription();

   public BufferedWriter report(OutputStream os) throws IOException {
      BufferedWriter oswriter = new BufferedWriter(new OutputStreamWriter(os));
      String s="Probe: " + this.getClass().getName()+"\n";
      s+=testDescription();

      if (!testDone)       s+="Test NOT DONE yet (execute test() method)\n";
      else if (testError) s+=TESTERROR+" performing test:
    "+testErrorMessage+"\n";
      else { //(testDone && !testError)
         if (compare()) s+=TESTPASSED+"\n";
         else        s+=TESTFAILLED+"\n";
      }
      oswriter.write(s);
      oswriter.flush();
      return oswriter;
   }

};
```

LISTING 5.2: TestProbe

5.1.1.3 TestCase

```
package probeAPI.probes;
```

```
import java.io.Serializable;

//Must implement serializable so that can be stored to file
public class TestCase implements Serializable {
   private static final long serialVersionUID = 1L;
   protected String in, out;

   public TestCase(String in, String out) {
         this.in = in;
         this.out= out;
      }
   public String in()   { return in; }
   public String out()  { return out; }
}
```

LISTING 5.3: TestCase

TestProbe class implements Serialisable this class serves as the serialisable object that will be sent or grouped to other objects. To accomplish this it is necessary to add a new class or method as a probe. TestProbe comes with a bunch of mechanisms that are used in constructing response messages in the report done after test execution.

TestCase class also implements Serialisable as it was said before to make the transportation of object of the probe be possible. TestCase serves more for input strings and output to be risen in at any moment.

5.1.2 Case study 1 - Network probe

The first probe consists of a testing of reachability of a host, to be more precise with a purpose of testing the internet connection and if the given address of some server exists and if it is turned on, then there will be a response. Thus, the testing probe input/output file will consist of internet addresses and the response should not be "Destination host unreachable" or "Unknown host". After creating this text file and using it in the probe class, the text file will be transformed into the .dat file, with a purpose to be readable not by the user but only by the .jar sent to him which will be executed during the process of testing.

The network probe is essential to guarantee that the PC for testing has its internet connection established since the whole system work is done via network. If user knows that his internet connection can fail, he must not contribute his computer for tests. Otherwise since it is used a socket, and the connection will suddenly be lost, socket will close and the test will fail. To prevent this situation on the web-server side there was implemented a for cycle where it is constantly searching for new socket connection, if it ended to the last possible port - 65535, that means there is no computers available and the web-server will act as testing environment, which resolve internet connection problem.

Next it is demonstrated the network probe hierarchy that consists of 3 classes:

5.1.2.1 TestProbeNet

```
package probeAPI.probes.net;

import probeAPI.probes.TestCase;
import probeAPI.probes.TestProbe;


abstract public class TestProbeNet extends TestProbe {
   private static final long serialVersionUID = 1L;
   protected String server;
   protected int port;

   public TestProbeNet(String server, int port, TestCase testcase) {
      super (testcase);
      this.server=server;
      this.port=port;
   }
```

```
};
```

Listing 5.4: TestProbeNet

5.1.2.2 TestProbePing

```
package probeAPI.probes.net;

import java.io.IOException;
import java.net.InetAddress;


public class TestProbePing extends TestProbeNet {
   private static final long serialVersionUID = 1L;
   private static final int defaultTimeout=5000; //Default ping wait (the
    test) timeout in milisecs (5 secs)

   private int timeout;
   private boolean reachable=false;

   public TestProbePing(String server) {
      this(server, defaultTimeout);
   }

   public TestProbePing(String server, int timeout) {
      super(server, 0, null);
      this.timeout=timeout;
   }

   public boolean compare() { return reachable; }

   //reachable is true if ping or echo (port 7) before timeout
   //false if not
   //if network error throws IOException and testError=true otherwise
    testError=false
   public void test() {
      try {
         InetAddress address = InetAddress.getByName(server);

         // Try to reach the specified address within the timeout
         // period. If during this period the address cannot be
         // reached then the method returns false.
         reachable = address.isReachable(timeout);
         testOK();
      } catch (IOException e) { testError(e); }
   }

   public String testDescription() {
```

```
      return "ping "+server+"\n";
   }

}
```

LISTING 5.5: TestProbePing

5.1.2.3 TestProbeSocket

```
package probeAPI.probes.net;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;

import probeAPI.probes.TestCase;

public class TestProbeSocket extends TestProbeNet {
   private static final long serialVersionUID = 1L;
   protected String serverReply;

   public TestProbeSocket(String server, int port, TestCase testcase) {
      super(server, port, testcase);
   }

   public void test() {
      //send data and get back from socket
      //NOTE: data is transferred using default charset encoding (What if
    encoding differs in server?)
      //if network error throws IOException and testError=true otherwise
    testError=false
      try {
         Socket socket = new Socket(server, port);
         PrintWriter    toServer;
         BufferedReader fromServer;

         toServer = new PrintWriter(socket.getOutputStream(), true);
          fromServer = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
```

```
            toServer.write(testcase.in());

            serverReply=fromServer.readLine();
            testOK();
         } catch (IOException e) { testError(e); }

      }

      public boolean compare() {
         return testcase.out().equals(serverReply);
      }

      public String testDescription() {
         return "Service: "+server+":"+port+"\n";
      }

      public BufferedWriter report(OutputStream os) throws IOException {
         BufferedWriter oswriter=super.report(os);

         if (testDone && !testError) {
            String s="Input:\n\""+testcase.in()+"\"\n";
            s+="Expected Output:\n\""+testcase.out()+"\"\n";
            s+="Output:\n\""+serverReply+"\"";
            oswriter.write(s);
         }
         oswriter.flush();
         return oswriter;
      }
}
```

Listing 5.6: TestProbeSocket

Each of this classes are subsequently related, TestProbePing and TestProbeSocket extend TestProbeNet, this TestProbeNet consists of 2 variables - Server (String) and port (Integer), TestProbePing class makes a ping test to the Server, TestProbeSocket class tries to make a socket connection and then sends data through it and waits back for the same data from socket.

5.1.3 Case study 2 - Userspace probe

The second probe is in userspace, this probe launches a shell Linux system or command prompt in windows, and can be used for testing basic commands like - "cd, ls / dir", or launching an executable made by the programmer. This executable could accept some input and the result will be certain output.

This probe also consists of 3 classes:

5.1.3.1 TestProbeUserSpace

```
package probeAPI.probes.userspace;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStream;

import probeAPI.probes.TestCase;
import probeAPI.probes.TestProbe;
import probeAPI.utils.LaunchUserSpaceProcess;

abstract public class TestProbeUserSpace extends TestProbe {
   private static final long serialVersionUID = 1L;
   protected String executablePath;
   protected String executableArguments;
   protected String stdout, stderr;

   public TestProbeUserSpace(String execpath, String execargs, TestCase
    testcase) {
      super (testcase);
      this.executablePath=execpath;
      this.executableArguments=execargs;
   }

   //if I/O error throws IOException and testError=true otherwise
    testError=false
   public void test() {
      try {
         LaunchUserSpaceProcess shell;
         shell = new LaunchUserSpaceProcess(executablePath,
    executableArguments, testcase.in());
         shell.run();
         stdout=shell.stdout();
```

```
            stderr=shell.stderr();
            testOK();
         } catch (Exception e) { testError(e); }
      }

      public String testDescription() {
         return "File: "+executablePath+""+executableArguments+"\n";
      }

      public BufferedWriter report(OutputStream os) throws IOException {
         BufferedWriter oswriter=super.report(os);

         if (testDone && !testError) {
            String s="Input:\n\""+testcase.in()+"\"\n";
            s+="Expected Output:\n\""+testcase.out()+"\"\n";
            s+="Output:\n\""+stdout+"\"\n";
            s+="stderr:\n\""+stderr+"\"\n";
            oswriter.write(s);
         }
         oswriter.flush();
         return oswriter;
      }
}
```

LISTING 5.7: TestProbeUserSpace

5.1.3.2 TestProbeUSstdinout

```
package probeAPI.probes.userspace;

import probeAPI.probes.TestCase;

public class TestProbeUSstdinout extends TestProbeUserSpace {
   private static final long serialVersionUID = 1L;

   public TestProbeUSstdinout(String execpath, String execargs, TestCase
    testcase) {
      super(execpath, execargs, testcase);
   }

   public boolean compare() {
      String stdouterr = stdout+stderr;
      return testcase.out().equals(stdouterr);
```

```
   }

}
```

LISTING 5.8: TestProbeUSstdinout

5.1.3.3 TestProbeUserSpace

```
package probeAPI.probes.userspace;

import probeAPI.probes.TestCase;

public class TestProbeUSstdinerr extends TestProbeUserSpace {
   private static final long serialVersionUID = 1L;

   public TestProbeUSstdinerr(String execpath, String execargs, TestCase
    testcase) {
      super(execpath, execargs, testcase);
   }

   public boolean compare() {
      return testcase.out().equals(stderr);
   }

}
```

LISTING 5.9: TestProbeUserSpace

The choice for which operative system that should do the test will be taken after creating LaunchUserSpaceProcess object in Userspace class that is constructed in LaunchUserSpaceProcess class, here the class OStype is called and the method getProperty("os.name") returns actual operative system name as a string, since there exist a vast variety of operative systems, the contributing user (recalling from chapter 4) will have no problem with any operative system, since the code will be tested on his computer. Without contribution, the testing environment is prepared as itself to make tests for 2 operative systems (during an experimental procedure), but this doesn't mean that there is no possibility of launching another computer with the operating system in fault. The ease of plug and play property facilitates in adding a new computer to the network of the testing environment.

5.2 The use of the testing environment to aid computing students exercises

For the purpose of real testing of the environment, there were used these two case studies, where some students from electronic and telecommunication engineering course participated, and also the explanation of the purpose of this tests were described on a DEEI's week seminar. The contribution was helpful to determine if the system was able to "monitor" many connections at the same time, and what is the limit where it stops working completely. All students that have participated were familiarised with Mooshak system, that is used in university as judge testing environment during courses like Imperative Programming and Object Oriented Programming, having so a kind of backup if they wanted to compare these 2 systems.

Students performed the tests successfully and these tests were done with an objective to see all the environment in stress, and obviously to accomplish this it was chosen 2 simple user space case studies. The amount of students involved were above 10 and the system performed well until the Fibonacci problem was initiated. Initial tests consisted from detecting operative system, detecting if the internet connection was stable, detecting if access was granted as super user, execution of command - expr 1 + 1 which will give an output of 2(in linux). The more complex test consisted in mapping first 50 Fibonacci numbers where the users upload their executable to the web server, and immediately after this upload is initiated, it is made a socket link with the secondary computer that is performing black box tests. There were written 2 types of programs - the first one that consisted of a "for" loop and second one consisted from recursive function.

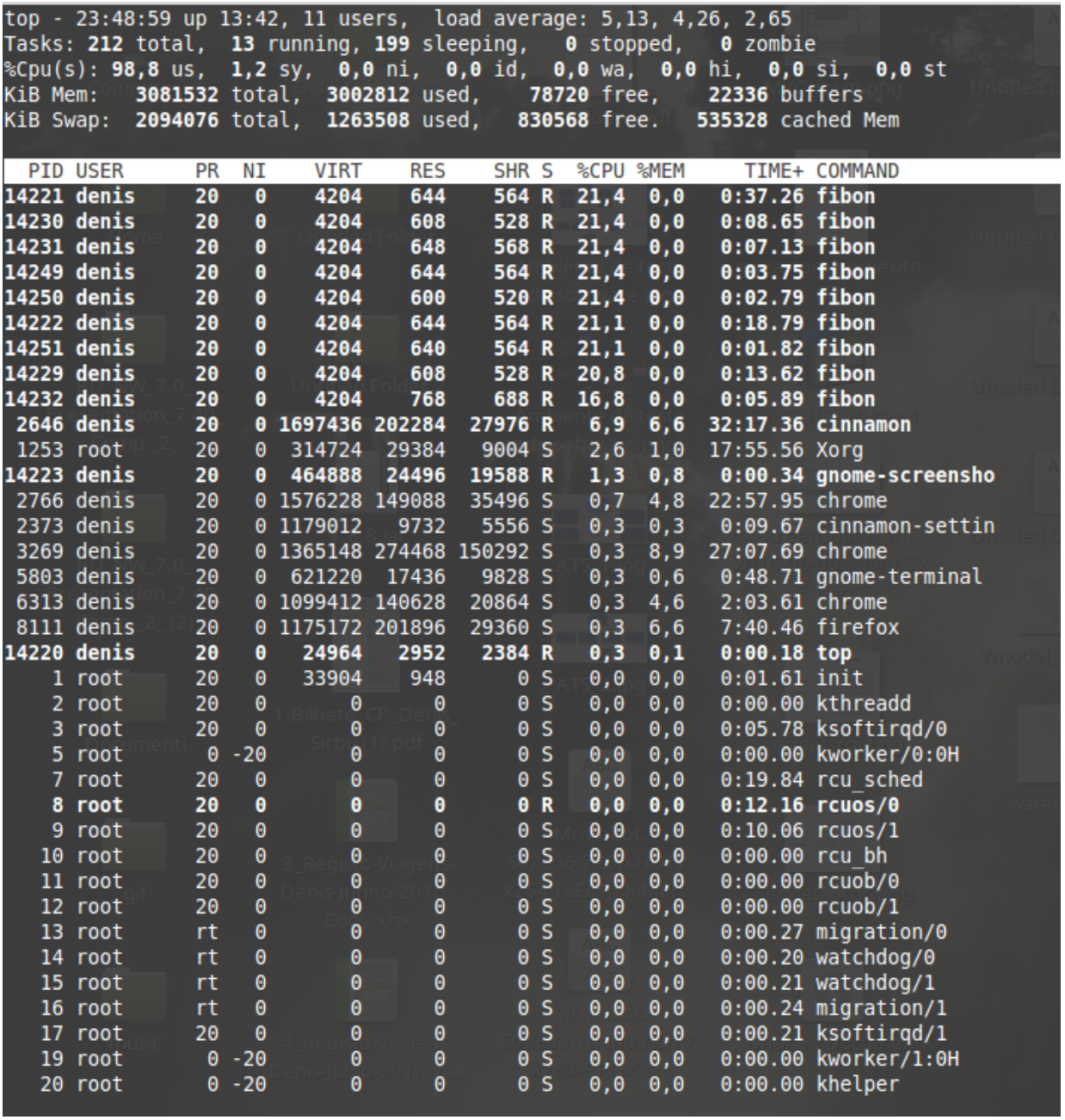


FIGURE 5.2: Fibonacci practical experience.

As you can see from figure 5.2 there is no possibility to identify which of processes are using a loop and which of them use a recursive function, this is only possible by analysing the source code which was not present in the uploaded file. Anyway adding more and more threads in java, the processing unit distributes them between all of the subsequent clients that are starting new tests, making so a division of CPU to be fair. Limiting java virtual machine to use 50% of computers core will guarantee that this computer will never become unstable. Since it was a stress test there was hidden the execution time parameter of any of programs but for safeness of the environment, it should be always added this parameter which is maximum executing time.

After the successful accomplishment of the stress test of the environment, it was suggested to see how is the effectiveness and the maximum limit of users that this tomcat server can handle with. The way how this could be made is making a script with a loop where each time it is done a get HTTP request. So researching more, there were found many solutions such as: Apache Benchmark [37], The Grinder [38], Gatling [39], FunkLoad [40], Jmeter and many others. For this problem and one of them that I thought would be more related to this work, and the most interesting in my opinion was the Jmeter. According to [41] this application designed to load test functional behaviour and measure performance.

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes	Latency	Connect Time(...
72	01:05:54.126	Thread Group ...	HTTP Request	737		1219	737	183
73	01:05:54.131	Thread Group ...	HTTP Request	736		1219	736	177
74	01:05:54.275	Thread Group ...	HTTP Request	598		1219	598	54
75	01:05:54.126	Thread Group ...	HTTP Request	749		1219	748	186
76	01:05:54.130	Thread Group ...	HTTP Request	746		1219	746	202
77	01:05:54.135	Thread Group ...	HTTP Request	743		1219	743	168
78	01:05:54.129	Thread Group ...	HTTP Request	753		1219	753	203
79	01:05:54.128	Thread Group ...	HTTP Request	754		1219	754	180
80	01:05:54.131	Thread Group ...	HTTP Request	754		1219	754	201
81	01:05:54.121	Thread Group ...	HTTP Request	765		1219	765	209
82	01:05:54.122	Thread Group ...	HTTP Request	768		1219	768	208
83	01:05:54.127	Thread Group ...	HTTP Request	764		1219	764	181
84	01:05:54.131	Thread Group ...	HTTP Request	764		1219	764	178
85	01:05:54.263	Thread Group ...	HTTP Request	636		1219	636	46
86	01:05:54.129	Thread Group ...	HTTP Request	772		1219	772	204
87	01:05:54.167	Thread Group ...	HTTP Request	737		1219	737	160
88	01:05:54.126	Thread Group ...	HTTP Request	779		1219	779	203
89	01:05:54.126	Thread Group ...	HTTP Request	781		1219	781	208
90	01:05:54.240	Thread Group ...	HTTP Request	669		1219	669	88
91	01:05:54.230	Thread Group ...	HTTP Request	681		1219	681	78
92	01:05:54.126	Thread Group ...	HTTP Request	788		1219	788	187
93	01:05:54.131	Thread Group ...	HTTP Request	785		1219	785	195
94	01:05:54.124	Thread Group ...	HTTP Request	793		1219	792	185
95	01:05:54.127	Thread Group ...	HTTP Request	790		1219	790	181
96	01:05:54.299	Thread Group ...	HTTP Request	620		1219	620	5
97	01:05:54.778	Thread Group ...	HTTP Request	242		1219	242	0
98	01:05:54.445	Thread Group ...	HTTP Request	690		1219	690	0
99	01:05:54.476	Thread Group ...	HTTP Request	660		1219	660	2
100	01:05:54.465	Thread Group ...	HTTP Request	671		1219	671	2

Scroll automatically? Child samples? No of Samples 100 Latest Sample 671 Average 430 Deviation 250

FIGURE 5.3: Test of logging to greetings page (29 of 100 get requests).

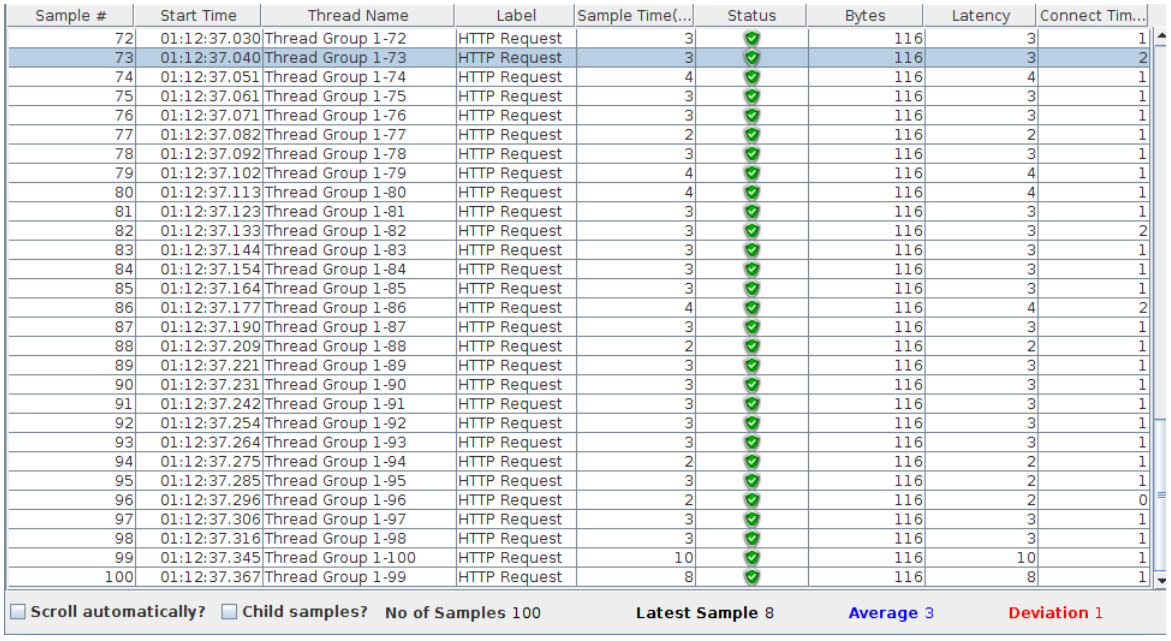
Looking to the figure 5.3 there are exemplified 29 of 100 samples of get HTTP request which consist from generating of the first page or greetings page on client side computer. This means that there will be a data transfer done from server to client. The total amount of bytes received on "client side" are 1219 where part of it is an image and the rest is rough text.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency	Connect Ti...
572	01:20:46.859	Thread Group 1-573	HTTP Request	7		1219	7	1
573	01:20:46.864	Thread Group 1-576	HTTP Request	6		1219	6	1
574	01:20:46.863	Thread Group 1-575	HTTP Request	8		1219	8	2
575	01:20:46.861	Thread Group 1-574	HTTP Request	9		1219	9	4
576	01:20:46.868	Thread Group 1-578	HTTP Request	5		1219	5	2
577	01:20:46.866	Thread Group 1-577	HTTP Request	8		1219	8	1
578	01:20:46.854	Thread Group 1-570	HTTP Request	28		1219	28	1
579	01:20:46.870	Thread Group 1-579	HTTP Request	13		1219	13	0
580	01:20:46.872	Thread Group 1-580	HTTP Request	11		1219	11	0
581	01:20:46.873	Thread Group 1-581	HTTP Request	11		1219	11	2
582	01:20:46.875	Thread Group 1-582	HTTP Request	12		1219	12	1
583	01:20:46.877	Thread Group 1-583	HTTP Request	12		1219	12	1
584	01:20:46.878	Thread Group 1-584	HTTP Request	15		1219	15	1
585	01:20:46.881	Thread Group 1-585	HTTP Request	12		1219	12	3
586	01:20:46.883	Thread Group 1-586	HTTP Request	16		1219	16	0
587	01:20:46.884	Thread Group 1-587	HTTP Request	15		1219	15	0
588	01:20:46.886	Thread Group 1-588	HTTP Request	14		1219	14	3
589	01:20:46.888	Thread Group 1-589	HTTP Request	12		1219	12	2
590	01:20:46.892	Thread Group 1-591	HTTP Request	11		1219	11	0
591	01:20:46.895	Thread Group 1-593	HTTP Request	10		1219	10	3
592	01:20:46.897	Thread Group 1-594	HTTP Request	10		1219	10	1
593	01:20:46.903	Thread Group 1-597	HTTP Request	7		1219	7	1
594	01:20:46.906	Thread Group 1-599	HTTP Request	4		1219	4	3
595	01:20:46.901	Thread Group 1-596	HTTP Request	10		1219	10	3
596	01:20:46.894	Thread Group 1-592	HTTP Request	19		1219	19	1
597	01:20:46.900	Thread Group 1-595	HTTP Request	14		1219	14	3
598	01:20:46.889	Thread Group 1-590	HTTP Request	26		1219	26	1
599	01:20:46.909	Thread Group 1-600	HTTP Request	8		1219	8	1
600	01:20:46.905	Thread Group 1-598	HTTP Request	31		1219	31	14

Scroll automatically? Child samples? No of Samples 600 Latest Sample 31 Average 24 Deviation 30

FIGURE 5.4: Test of logging to greetings page(29 of 600 get requests).

If we compare 2 figures 5.3 and 5.4 we can see that with 600 get requests the average latency is smaller, and the reason must be because of some process that was launched at same time recently, that should have been using data transfer as well, so the server was not able to respond effectively.



Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes	Latency	Connect Tim...
72	01:12:37.030	Thread Group 1-72	HTTP Request	3		116	3	1
73	01:12:37.040	Thread Group 1-73	HTTP Request	3		116	3	2
74	01:12:37.051	Thread Group 1-74	HTTP Request	4		116	4	1
75	01:12:37.061	Thread Group 1-75	HTTP Request	3		116	3	1
76	01:12:37.071	Thread Group 1-76	HTTP Request	3		116	3	1
77	01:12:37.082	Thread Group 1-77	HTTP Request	2		116	2	1
78	01:12:37.092	Thread Group 1-78	HTTP Request	3		116	3	1
79	01:12:37.102	Thread Group 1-79	HTTP Request	4		116	4	1
80	01:12:37.113	Thread Group 1-80	HTTP Request	4		116	4	1
81	01:12:37.123	Thread Group 1-81	HTTP Request	3		116	3	1
82	01:12:37.133	Thread Group 1-82	HTTP Request	3		116	3	2
83	01:12:37.144	Thread Group 1-83	HTTP Request	3		116	3	1
84	01:12:37.154	Thread Group 1-84	HTTP Request	3		116	3	1
85	01:12:37.164	Thread Group 1-85	HTTP Request	3		116	3	1
86	01:12:37.177	Thread Group 1-86	HTTP Request	4		116	4	2
87	01:12:37.190	Thread Group 1-87	HTTP Request	3		116	3	1
88	01:12:37.209	Thread Group 1-88	HTTP Request	2		116	2	1
89	01:12:37.221	Thread Group 1-89	HTTP Request	3		116	3	1
90	01:12:37.231	Thread Group 1-90	HTTP Request	3		116	3	1
91	01:12:37.242	Thread Group 1-91	HTTP Request	3		116	3	1
92	01:12:37.254	Thread Group 1-92	HTTP Request	3		116	3	1
93	01:12:37.264	Thread Group 1-93	HTTP Request	3		116	3	1
94	01:12:37.275	Thread Group 1-94	HTTP Request	2		116	2	1
95	01:12:37.285	Thread Group 1-95	HTTP Request	3		116	2	1
96	01:12:37.296	Thread Group 1-96	HTTP Request	2		116	2	0
97	01:12:37.306	Thread Group 1-97	HTTP Request	3		116	3	1
98	01:12:37.316	Thread Group 1-98	HTTP Request	3		116	3	1
99	01:12:37.345	Thread Group 1-100	HTTP Request	10		116	10	1
100	01:12:37.367	Thread Group 1-99	HTTP Request	8		116	8	1

Scroll automatically? Child samples? No of Samples 100 Latest Sample 8 Average 3 Deviation 1

FIGURE 5.5: Test of logging into server(29 of 100 post requests).

After the post requests the average latency became incredibly smaller than during the get requests, possibly this happened because of the amount information. At this time there are only 116 bytes of data that the server has sent to the Jmeter.

This 4 images 5.3, 5.4, 5.5, 5.6 from the moment when there is a small amount of data to be received or sent, and comparing get and post request proves this in latency. It also should be mentioned that the Jmeter was not executed on the same computer where the web server is installed and all tests were executed from 192.168.1.66 to 192.168.1.76 and the communication was done through router rj45 cable with maximum transfer velocity of 1Gbps. All requests that were produced during the execution of Jmeter were in ramp-up interval of 1 second.

Computer specs:

- CPU Intel core 2 Duo T7300: 2x2.0GHz (4Mb Cache)
- GPU ATI Mobility Radeon X2500
- Ram DDR2 2GB
- HDD 120GB 7200 r.p.m Western Digital

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes	Latency	Connect Tim...
572	01:15:10.136	Thread Group 1-564	HTTP Request	30		116	30	5
573	01:15:10.162	Thread Group 1-577	HTTP Request	4		116	4	2
574	01:15:10.163	Thread Group 1-578	HTTP Request	3		116	3	1
575	01:15:10.159	Thread Group 1-576	HTTP Request	7		116	7	1
576	01:15:10.145	Thread Group 1-569	HTTP Request	22		116	22	1
577	01:15:10.130	Thread Group 1-561	HTTP Request	37		116	37	1
578	01:15:10.133	Thread Group 1-562	HTTP Request	34		116	33	4
579	01:15:10.460	Thread Group 1-580	HTTP Request	4		116	4	1
580	01:15:10.461	Thread Group 1-581	HTTP Request	4		116	4	1
581	01:15:10.458	Thread Group 1-579	HTTP Request	8		116	8	1
582	01:15:10.466	Thread Group 1-583	HTTP Request	3		116	3	2
583	01:15:10.467	Thread Group 1-584	HTTP Request	2		116	2	0
584	01:15:10.463	Thread Group 1-582	HTTP Request	6		116	6	1
585	01:15:10.472	Thread Group 1-585	HTTP Request	3		116	3	1
586	01:15:10.483	Thread Group 1-587	HTTP Request	2		116	2	0
587	01:15:10.488	Thread Group 1-590	HTTP Request	6		116	6	0
588	01:15:10.487	Thread Group 1-589	HTTP Request	8		116	8	3
589	01:15:10.486	Thread Group 1-588	HTTP Request	10		116	10	6
590	01:15:10.492	Thread Group 1-591	HTTP Request	5		116	4	2
591	01:15:10.481	Thread Group 1-586	HTTP Request	16		116	16	3
592	01:15:10.561	Thread Group 1-593	HTTP Request	4		116	4	2
593	01:15:10.563	Thread Group 1-594	HTTP Request	5		116	5	1
594	01:15:10.567	Thread Group 1-595	HTTP Request	2		116	2	0
595	01:15:10.566	Thread Group 1-592	HTTP Request	4		116	4	1
596	01:15:10.579	Thread Group 1-597	HTTP Request	3		116	2	1
597	01:15:10.583	Thread Group 1-598	HTTP Request	2		116	2	1
598	01:15:10.586	Thread Group 1-599	HTTP Request	3		116	3	1
599	01:15:10.588	Thread Group 1-600	HTTP Request	3		116	3	1
600	01:15:10.594	Thread Group 1-596	HTTP Request	2		116	2	1

Scroll automatically? Child samples? No of Samples 600 Latest Sample 2 Average 17 Deviation 24

FIGURE 5.6: Test of logging into server(29 of 600 post requests).

- Network - Wireless card type N and Realtek PCIe FE

To make the difference between 2 systems we have seen already that there is a possibility of user contribution to make assurance of all of the tests. This assurance is a kind of redirecting page in the case if the user suddenly decides to change his mind in this "contribution" or in case he needs to leave or in the case when the internet connection is lost.

Both systems from this project and the Mooshak were explained previously and was also seen that all architecture is very different from each, while user view may have some similarities, for example, in both systems user should upload a file and a report is shown after the test. In Mooshak the report is not so easy to understand, to detect where the error is made you should have access to the server, while in my environment the user sees perfectly the case where the error occurred, which clarifying the complete error case.

Chapter 6

Conclusions and future work

6.1 Recalling the initial part of the work

As a conclusion for this work it should be mentioned the importance of testing in software production, since the moment when the software life cycle is modelled should exist always a testing operation done to not expect same errors as they were done from historically known failures.

To try to formulate again what is software life cycle - it is the period of time beginning with the appearance of the concept of software which ends when the use of the software is no longer possible. That life cycle was described in Introduction (chapter 1) and it includes the following stages: requirement analysis, software design, software developing, unit testing, white box testing, black box testing, beta testing, release, support, end-of-life.

Testing parts described in life cycle depends on exactly from the purpose of the software as itself. And again, recalling from chapter 1 - the V-shaped software life cycle, as the demonstration, in which stages the software environment can be used, or better - stages like acceptance test and unit test.

In the chapter 2 the objective was to introduce the reader to the creation of the testing environment and what necessities that were needed to launch it. Also was described the full concept of how it was done and what programming languages were used to create the web-server side of the environment. With the help of diagrams was demonstrated a testing scenario of connection to the system.

In the chapter 3 was modelled a design pattern of the projection of the system, and it was described in more details how the concept of communication between web server

and testing machine works. Then it was demonstrated how to use libraries in java EE to communicate with help of internet protocols.

In the chapter 4 was demonstrated an interaction of users with the environment and the exemplification of a user contribution to a system.

In the chapter 5 was made an exemplification of the test probes and also, there were performed some stability tests of the web server, and stress test on the testing system. and the last chapter is the chapter of conclusions and future work.

During the construction of this system, it has been studied the process of software testing used nowadays all other the world. There were considered most types of defects in the software programming. It was also studied how to create and use the test cases for the black box. This material can be used for educational purposes and not only, since the main references were taken from papers and books from institutions like IEEE and other universities and companies.

- it was created a testing environment that is able to adapt to the API of programmable probes
- it was introduced an explanation of how the distributed architecture that is client-server type where the server part has similarities with master-slave concept which acts during the process

6.2 Future work

For future reworks, it can be added a database that will provide a faster response when logging into the system, and a database that treats the part of the report will help in managing the response given from server.

To ensure that users don't use some malicious software instead of the real executable, there can be prepared some virtual machines that will act as protection, and every time if something goes wrong the system could be rebooted or be invoked with some previously saved emulation of operative system.

There also should be a rework of the design of the web server, because now it has a very simple look. This could be accomplished by adding some CSS and javascript.

One more detail that could be added is more competition among users for example if it would be created a programming context where the objective is to resolve some problem like in mooshak happens, and every user can see the successful or unsuccessful attempts

of others, creating so a kind of "rivalry" between each other, what will make users not to give up so quickly.

6.3 Conclusion

The help of that software testing environment is important to all audience that has in mind to construct a concrete program, which is at the same time a project with some kind of objective. The use of this environment will make the user to considerably prevent all sort of errors that are present now but never forgetting that the project manager is the person who should configure all of the tests. If the program is the case of at least a group of functions that should work together and after combining them will provide no integrity errors, the software testing probes could be prepared to see if the tests that were done previously have passed with success.

To conclude the complete work, the system works in a next manner - when there is a user or many users that want to test their code, they log into the website and they are able to choose to be a part of this testing environment or not, providing so some help with their computer. When a user considers this, there is also a possibility to download the full version of probes or single probe that will be chosen by him next(full version means that other users can make tests on his computer too, while single is just for his probe), after that he uploads the executable file to the web server with an objective of gathering so a copy of the program on the server, it is created at same time a file with username and log.txt in it. After the successful upload the user with single contribution downloads a .jar consisting of the probe with .dat file (all zipped into one), then, after extraction, he should execute this .jar file. At this moment all tests are started and after the finish, it is made a socket connection to the server and the results are written into the log file on the web server, which reads the data from this file and if there will be any difference the result is provided to the user in the web browser. If the user helps us with the full contribution, the socket connection is done immediately after executing the .jar file and all new users if there is no more testing machines would connect to it to test their program. Imagine if it would be added 4 computers with full contribution level, and the socket connection is done from ports 4444 to 4447. Web server tries to send executable to 4444 if it is busy it passes to 4445 and so on until it finds one available. But imagine if all computers are busy then there is a possibility of launching the test on the own web server, but this is not secure, and almost impossible to happen, since the .jar file has implemented in it threading and every time a test is started it is created a new process with another socket.

Continuing the explanation, this environment becomes fully automated after all probes are constructed and created and it is a good tool for program managers and team leaders to inspire their colleagues to prevent errors that could appear after the release of the program.

According to the state of arts resources, the differences between last 20 years are not so big when speaking about software testing and the main concepts remain the same, which are the black box and white box testing, what makes this work to remain updated.

Considering user interaction made and explained in chapter 4 and the practical part realised in that chapter we can conclude that the environment performs better when the user uses it's own computer as a testing machine, and this is reasonably understandable. It is so because all the computation process is done on the user's computer. The web server in this case just confirms the output that is produced on client side machine, and unique limit that was possible to see is the limit of the speed of the connection. Since the router which connects all the computers accepts, imagine a 1 gigabit of data transfer, and if we need to send or receive 2500 bytes of data we have the next equations:

$$2.500\ byte\ *\ 8\ bit\ =\ 20.000\ bit\ (1\ byte\ =\ 8\ bit) \tag{6.1}$$

$$1.000.000\ bit\ :\ 20.000\ =\ 50 \tag{6.2}$$

According to 6.1 and 6.2 theoretically it is possible to have 50 connections at the same time, which is equivalent to 50 users that will connect to the web server and all of them will upload data, with instantaneous response. The latency values observed in 6 proved that even with 600 users the system responds great and the packet received during that test was of 1219 bytes, making again some calculations:

$$1.219\ byte\ *\ 8\ bit\ =\ 9.752\ bit \tag{6.3}$$

$$1.000.000\ :\ 9.752\ =\ 102,54 \approx 103 \tag{6.4}$$

we have the equation 6.4 that explains that after all 103 users that make connection simultaneously and after all of them doing requests of the data from server it should have an immediate response, while if we add more users the latency will be considerably slower, this is possible to see from the figure 6.1 that the average latency from 103 requests is of 2ms while 618 requests have a latency of 36ms and a huge deviation.

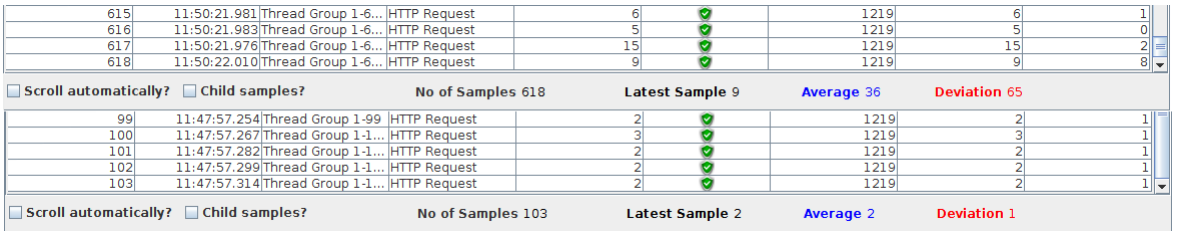


FIGURE 6.1: Comparison between 2 samples. At the right side we have the last 4 of 103 requests made to server simultaneously, while on left side we have the last 4 of 618 requests

All this data was gathered with the help of software - Jmeter [41]. With the data provided during the tests that were performed in 5 in section 5.2 it is possible to realise that the tomcat server performs well as web server even with enormous amount of users that are trying to make the connection to it, but it certainly depends on the amount of information that is travelling from server to client and figures from this and previous section confirm this statement. After the stress test that was done where some students from my course participated it should be mentioned that in java virtual machine settings we can limit the amount of CPU that is necessary for the execution of any kind of test. That could be prepared by the tester and doing so will prevent the system to crash during the tests that were prepared, but the best way to avoid this is to use the contribution of the user's computer.

Testing some infinite loop cases like was demonstrated with fibonacci problem we could see that threading system distribution makes a fair division of processing amount and this is clear that the tests would never stop from the moment that there is sufficient amount of RAM available and system will not stop the "parent" process to avoid computer congestion.

By finishing this work I should add that all this document was fully written using LaTeX [42], which was firstly implemented in Texmaker program, and after, I swapped the Texmaker program with web platform - overleaf [1]. Overleaf is an on-line solution to the Texmaker and it provides with the possibility to write anywhere where you have the internet access. The reason why I choose Latex as document writing tool is because I did not want to lose time by inventing templates for Word and also because latex is free and previously I already had a lot experience with it, and like the LaTeXslogan says - "It is better to leave document design to document designers, and to let authors get on with writing documents".

[1] For more information about overleaf project consult [43]

Bibliography

[1] Bruce Eckel. *Thinking in Java.* Pentince Hall., 2007.

[2] Yogesh Singh. *Software testing.* CAMBRIDGE UNIVERSITY PRESS, 2012.

[3] Prof. J. L. LIONS. ARIANE 5 - Flight 501 Failure. http://www.math.umn.edu/~arnold/disasters/ariane5rep.html. [Online; last accessed 17/10/2016].

[4] BBC News World Edition. Y2K bug fails to bite. http://news.bbc.co.uk/2/hi/science/nature/585013.stm. [Online; last accessed 17/10/2016].

[5] Douglas N. Arnold. The Patriot Missile Failure. http://www.math.umn.edu/~arnold/disasters/patriot.html. [Online; last accessed 17/10/2016].

[6] Ross J. Anderson. *Information Technology in Medical Practice: Safety and Privacy Lessons from the United Kingdom.* University of Cambridge Computer Laboratory, 1998.

[7] Anthony Finkelstein. *Report of the Inquiry into the London Ambulance Service.* University College, London, 1993.

[8] R.N. Charette. *Why Software Fails.* IEEE Spectrum, 2005.

[9] Software & Systems Engineering Standards Committee. IEEE 610 standard. https://standards.ieee.org/findstds/standard/610.12-1990.html. [Online; last accessed 17/10/2016].

[10] José Paulo Leal. Mooshak is a system for managing programming contests. https://mooshak.dcc.fc.up.pt/ [Online; accessed 10-June-2015].

[11] Upcounsel, a marketplace for legal services. https://www.upcounsel.com/beta-test-agreement, . [Online; accessed 20-May-2015].

[12] Non-disclosure-agreement. http://www.ndasforfree.com/NDAS/GetSoftwareBeta.html, . [Online; accessed 20-May-2015].

[13] Cisco systems. http://www.cisco.com/public/beta-agreement.html, . [Online; accessed 20-May-2015].

[14] Angelina Samaroo Geoff Thompson Brian Hambling (Editor), Peter Morgan and Peter Williams. *SOFTWARE TESTING An ISTQB–ISEB Foundation Guide*. BCS - The Charted Institute of IT, 2010.

[15] В.В. Липаев. *Программная Инженерия, Методологические Основы*. Теис, 2006.

[16] Pierre Bourque, École de technologie supérieure (ÉTS), and Richard E. (Dick) Fairley. *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2014.

[17] Hans Schaefer Andreas Spillner, Tilo Linz. *Software Testing Foundations*. O'Reilly, 2014.

[18] Jose Paulo Leal and Fernando Silva. Managing programming contents with mooshak. https://www.dcc.fc.up.pt/~zp/papers/mooshak-spe.pdf, 2001. Online; last accessed 17/10/2016.

[19] TCL'ers. TCL. http://wiki.tcl.tk/299. [Online; last accessed 10-May-2015].

[20] Kurt Wall. *Tcl and Tk Programming for the Absolute Beginner*. Course Technology, 2009.

[21] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Professional, 2009.

[22] Ken Jones Brent Welch. *Practical Programming in Tcl and Tk*. Williams, 2004.

[23] Tjandra Satria Gunawan Mitch Schwartz, Jacob Plachta et al. About the SPOJ System. http://www.spoj.com/info/. [Online; last accessed 20/10/2016].

[24] Tojo Chacko Bhavin Turakhia, Basil Skariah. About CodeCheff. https://www.codechef.com/aboutus. [Online; last accessed 20/10/2016].

[25] Universidade Regional Integrada do Alto Uruguai e das Missões. THE URI ONLINE JUDGE. https://www.urionlinejudge.com.br/judge/en/login. [Online; last accessed 20/10/2016].

[26] International Organization for Standardization. ISO/IEC 25010:2011. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. [Online; last accessed 20/10/2016].

[27] Raspberry Pi Foundation. Raspbery Pi. https://www.raspberrypi.org/help/what-is-a-raspberry-pi/, . [Online; accessed 19-July-2015].

[28] Andy Oram. *Learning PHP, MySQL, and JavaScript*. O'Reilly, 2009.

[29] Ken O. Burtch. *Linux Shell Scripting With Bash*. Sams, 2004.

[30] The Apache Software Foundation. Apache. https://tomcat.apache.org/whoweare.html, . [Online; accessed 29-July-2015].

[31] Kevin Jones Jayson Falkner. *Servlets and JavaServer Pages. The J2EE Technology Web Tier*. Addison-Wesley, 2009.

[32] Berkley University IST. Model View Controller: A design pattern for software. https://ist.berkeley.edu/as-ag/pub/pdf/mvc-seminar.pdf. [Online; accessed 29-September-2015].

[33] Sun Microsystems. Interface HttpSession. https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpSession.html. [Online; accessed 29-September-2015].

[34] David Gourley, Sailu Reddy Brian Totty with Majoty Sayer, and Anshu Aggarwal. *HTTP: The Definitive Guide*. O'REILLY, 2002.

[35] Keith W. Ross James F. Kurose. *Computer Networking - A top down Approach*. Pearson, 2013.

[36] Daniel H.; M. M. M. Moura; Ana Leiria. Intensive software testing and evaluation on a grid. research, reflections and innovations in integrating ict in education: Proceedings book of the v international conference on multimedia and ict in education (m-icte2009). In *ISBN*, volume III, pages 1353–1357, Lisbon, Portugal, 2009. FORMATEX.

[37] Apache. ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/current/programs/ab.html. [Online; last accessed 21/10/2016].

[38] Philip Aston, Paco Gómez, and Peter Zadrozny. The Grinder, a Java Load Testing Framework. http://grinder.sourceforge.net/. [Online; last accessed 21/10/2016].

[39] Stéphane Landelle, Guillaume Corré, Flavien Bert, Thomas Grenier, and Paul-Henri Pillet. Gatling. http://gatling.io/docs/2.2.2/. [Online; last accessed 21/10/2016].

[40] Benoit Delbosc. FunkLoad. http://funkload.nuxeo.org/. [Online; last accessed 21/10/2016].

[41] The Apache Software Foundation. JMeter. http://jmeter.apache.org/, . [Online; accessed 28-October-2015].

[42] Donald Knuth. Latex. https://www.latex-project.org/about/. [Online; accessed 10-June-2015].

[43] John Hammersley and Mary Anne Baynes. Overleaf. https://www.overleaf.com/. [Online; accessed 16-August-2015].